Task Performance Measurement with Frame Rate Upconversion and Latency Compensation Using Image Based Rendering

Uku Hirvesoo

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Helsinki 25.11.2015

Thesis supervisor:

Asst Prof. Jaakko Lehtinen

Thesis advisor:

Adj. Prof. Jukka Häkkinen

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Uku Hirvesoo

Title: Task Performance Measurement with Frame Rate Upconversion and Latency Compensation Using Image Based Rendering

Date: 25.11.2015 Language: English Number of pages: 8+60

Department of Computer Science

Professorship: Computer graphics research

Supervisor: Asst Prof. Jaakko Lehtinen

Advisor: Adj. Prof. Jukka Häkkinen

Traditionally in computer graphics complex 3D scenes are represented as a collection of more primitive geometric surfaces. The geometric representation is then rendered into a 2D raster image suitable for display devices.
Image based rendering is an interesting addition to a geometry based rendering. Performance is constrained only by display resolution, and not by scene geometry complexity or shader complexity. When used together with a geometry based renderer, an image based renderer can extrapolate additional frames into an animation sequence based on geometrically rendered frames.
Existing research into image based rendering methods is investigated in context of interactive computer graphics. Also an image based renderer is implemented to run on a modern GPU shader architecture. Finally, it's used in a first person shooter game experiment to measure task performance when using frame rate upconversion.
Image based rendering is found to be promising for frame rate upconversion as well as for latency compensation. An implementation of an image based renderer is found feasible on modern GPUs. The experiment results show considerable improvement in test subject hit rates when using frame rate upconversion with latency compensation.

Keywords: 3D warping, image based rendering, reprojection, frame rate upconversion, latency compensation

AALTO-YLIOPISTO
PERUSTIETEIDEN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Uku Hirvesoo

Työn nimi: Koehenkilöiden suorituskykymittaukset: kuvataajuuden kasvattaminen ja latenssin kompensointi käyttäen kuvapohjaista renderöintia

Päivämäärä: 25.11.2015 Kieli: Englanti Sivumäärä: 8+60

Tietotekniikan laitos

Professuuri: Tietokonegrafiikan tutkimusryhmä

Työn valvoja: Asst Prof. Jaakko Lehtinen

Työn ohjaaja: Dosentti Jukka Häkkinen

Perinteisesti tietokonegrafiikassa monimutkaiset kolmiulotteiset maisemat kuvaillaan yksinkertaisempien geometristen pintojen kokoelmana. Geometrisesta kuvauksesta renderöidään kaksiulotteinen näyttöille sopiva rasterikuva.
Kuvapohjainen renderöinti on mielenkiintoinen lisäys geometriapohjaisen renderöinnin rinnalle. Suorituskyky ei riipu virtuaalimaiseman geometrisestä monimutkaisuudesta tai varjostustehosteiden raskaudesta, vaan ainoastaan näytön erottelukyvystä. Yhdessä geometriapohjaisen renderöinnin kanssa käytettynä kuvapohjainen renderöija voi ekstrapoloida uusia kuvia animaatiosekvenssiin vanhojen tavallisesti renderöitujen kuvien perusteella.
Kuvapohjaista renderöintia tutkitaan vuorovaikutteisen tietokonegrafiikan näkökulmasta olemassa olevan kirjallisuuden pohjalta. Lisäksi toteutetaan kuvapohjainen renderöija nykyaikaisille grafiikkasuorittimille. Lopuksi toteutetaan käyttäjäkoe käyttäen kuvapohjaista renderöijaa kuvataajuuden kasvattamiseksi, jossa koehenkilöiden suorituskykyä mitataan ammuskelupelissä.
Kuvapohjainen renderöinti todetaan lupaavaksi keinoksi kuvataajuuden kasvattamiseksi ja latenssin kompensointiin. Kuvapohjaisen renderöijan toteuttaminen nykyaikaiselle grafiikkasuorittimille todetaan mahdolliseksi. Käyttäjäkokeen tulokset osoittavat, että koehenkilöiden osumatarkkuus koheni merkittävästi kun käytettiin kuvataajuuden kasvattamista ja latenssin kompensointia.

Avainsanat: 3D vääntäminen, kuvapohjainen renderöinti, uudelleenprojisointi, kuvataajuuden kasvattaminen, latenssin kompensointi

Acknowledgments

I want to thank all the persons who have been influential in making this thesis possible.

Mikko Nuutinen taught the courses on Digital Imaging Technology and Visual Media. The papers I wrote for these courses led me to dealing with frame rate upconversion in the first place.

Jaakko Lehtinen, has provided me with valuable skills through his Interactive Computer Graphics course. Also as the supervising professor of this thesis, his help has been invaluable.

Jukka Häkkinen has helped me with designing the task performance test methodology. Also his personal interest in my project has helped me keep the goal in sight even during periods when progress was slow.

My Bachelor's thesis supervisors Mervi Ranta and Henrik Asplund taught me scientific writing, an experience I've relied on heavily during this project.

Finally both my spouse Anna and my dear friend Teemu deserve honorary notice for believing in me and supporting me through this tough process.

Helsinki, 14.11.2015

Uku Hirvesoo

Contents

Abbreviations

CRT	Cathode Ray Tube (display technology)
LC	Liquid Crystal (display technology)
LCD	Liquid Crystal Display (display using LCs)
LCD TV	Liquid Crystal Display Television
LED	Light Emitting Diode
PAL	Phase Alternating Line (color television standard)
NTSC	National Television System Committee (color television standard)
VGA	640x480 display resolution
1080p	1920x1080 display resolution
4K	display resolution that is roughly 4000 pixels wide, commonly 3840x2160 (twice the 1080p resolution)
HFR	High Frame Rate
GPU	Graphics Processing Unit
OpenGL	Open Graphics Library
SoC	System on a Chip
HUD	head-up display, an overlay for displaying information to user
GiB	Gibibyte, 1 GiB = 1024 * 1024 * 1024 bytes
ms	millisecond

Glossary

human visual system	A neurological system comprising eyes and parts of human brain which processes visual information
retina	A light sensitive surface within the eye onto which the visible world is projected
backlight	A light source inside LCD displays, behind the LC cells
frame	An image in an animation sequence
color buffer	A matrix of color values in memory holding a representation of an image
frame buffer	An operating system managed color buffer used for transfering images to the display device
Z-buffer	A matrix of depth values corresponding to the color values in color buffer
velocity buffer	A matrix of 3D velocity values corresponding to the color values in color buffer
pixel	A square area with single color value in the color buffer or the display device
color fragment	A temporary color sample from a point on a surface which may become a pixel if stored in a buffer
voxel	A 3D generalization of a pixel
voxel volume	A volume built out of voxels
tuple	A list of values stored in a single structure
refresh rate	The rate at which a display device can receive frames
frame rate	The rate at which animation source material can emit frames
interpolation	An act of construction new data points in between existing data points
extrapolation	An act of construction new data points after existing data points
frame rate upconversion	The act of inserting new frames into an animation sequence by means of interpolation or extrapolation from existing frames
color interpolation	The act of coloring a continuous surface based on known color samples from points on the surface
scene geometry	A set of geometric primitives, usually triangles, connected to each other to model surfaces of more complex objects
rendering	A process of generating images based on some other, often geometric, representation
shader program	A program which customizes how the GPU pipeline processes graphics data

1 Introduction

Motion blur is an inherent attribute of LCD displays, causing moving images to look blurred. High refresh rate televisions solve the motion blur problem by increasing the refresh rate. Since no high refresh rate source material is available, such televisions upconvert the existing low rate material with motion compensation techniques. Motion compensation interpolates new intermediary frames by estimating the movement of image areas from original frames.

As a keen computer gamer the idea of using similar techniques with live 3D computer graphics has occurred to me each time when dealing with the subject of motion blur in course papers. After all, with computer graphics, motion need not be estimated because the simulated virtual environment already has information for describing how objects move. Unfortunately I never had time to deal with the subject beyond noting the possibility as a future direction of research. This background formed a natural premise for this Master's thesis. In context of computer graphics motion compensation is called image based rendering instead.

Three goals as set for the thesis. First, a review of literary background should be done to find previous research on attempts to use image based rendering. Second, a simple live frame rate upconversion algorithm should be implemented to see if it's feasible on modern GPU. Third, the implemented algorithm should be used as part of a task performance experiment to see if frame rate upconversion translates into some tangible advantage.

1.1 Structure

First in section 2 some motivating applications are considered where image based rendering techniques could be useful in context of computer graphics. In section 3 human visual system is described and the important concept of hold-type motion blur is introduced. Section 4 deals with the first goal of performing literary study and other theoretical background. Section 5 documents the process of designing an application which implements a frame rate upconversion algorithm and the way it's used in a task performance experiment. As such it deals with second and the third goals. Section 6 presents the obtained results from the experiment. In section 7 the results are discussed and analyzed against the goals. Section 8 present conclusions and finally section 9 identifies unanswered question left for possible future research.

1.2 Excluded discussion

Some rendering techniques such as anti-aliasing and transparency blend multiple geometry surface samples onto each pixel in the color buffer. Due to the inherent ambiguity such techniques are difficult to combine with pixel reprojection (see subsection 4.2.2). Unless explicitly stated otherwise, it's assumed from now on that each pixel in a rasterized image maps onto a single surface point in the scene.

1.3 Contributions

First this thesis provides an overview of image based rendering techniques usable for frame rate upconversion.

Also a concrete implementation of a frame rate upconversion algorithm is developed, which is released as open-source (see appendix A). The most important piece of code is the OpenGL shader program that does the bulk of the work. The shader implements a novel tessellator based sparse grid optimization technique. The algorithm not only upconverts the frame rate, but also compensates for two kinds of latency: view change latency and scene change latency. As far as is known, scene change latency compensation has not been done before.

Finally this thesis contributes the task performance experiment results. It is proved that the developed algorithm in fact enhances task performance significantly in first person shooter type games.

2 Motivating applications

The motivation to investigate image based rendering for live 3D computer graphics can stem from a number of potential applications.

2.1 High Resolution and High Refresh Rate Displays

For a long time the LCD display industry has adopted a de-facto 60 Hz refresh rate. This set an effective upper bound to GPU temporal performance demand from consumers. Only recently has the LCD industry began making displays with support for higher refresh rates. Common new refresh rates are 120 Hz and 144 Hz, roughly double the existing 60 Hz.

More or less simultaneously with the introduction of high refresh rate LCD displays, a new "4K" resolution class is being introduced. The new "4K" displays are commonly defined to have 3840x2160 resolution, twice the current 1080p standard in both spatial dimensions. The pixel count is multiplied by four.

With VESA announcing the DisplayPort 1.3 standard [1], the media has widely interpreted [2] the bandwidth to be high enough to allow 4K resolution with 120 Hz. Comparing both the spatial and the temporal increases over 1080p with 60Hz, we end up with 8x more pixels that need to be rendered each second. While we have seen roughly annual incremental updates to GPU performance, 8x increase in rasterization, texturing, and fragment shading resources might be unachievable in the time frame it takes to see 120 Hz 4K displays on the market. Also, already there are announcements of 8K display appearing [3], which presumably further quadruples the pixel counts the near future GPUs are expected to deal with.

Such a market situation strongly motivates finding new ways of doing more with less resources. The higher the display refresh rates become, the higher the frame-to-frame coherence. With high amount of frame-to-frame coherence to take advantage of, frame-rate upconversion becomes very promising. Scene geometry may

not need to be rendered at an increased rate while still benefitting from motion blur reduction and low controller input latency.

Increases in spatial resolution, in principle, also burdens image based renders equally to geometry based renderers. However by using a sparse reconstruction grids, the reconstruction grid need not necessarily match the display resolution. It may be possible to scale image based renderer designs to higher resolution displays without growing the grid density accordingly. The use of sparse reconstruction grids in such a manner requires further research.

2.2 Virtual Reality

Interest in consumer class head mounted displays (HMD) is as of writing this a hot topic. Multiple manufacturers are currently actively developing products: Oculus Rift [4], Samsung Gear VR [5], Sony Project Morpheus [6], HTC Vive [7], Microsoft HoloLens [8], Fove [9], Zeiss VR One [10], Avegant Glyph [11], OSVR (Open Source Virtual Reality) Hacker Dev Kit [12], Google Cardboard [13], ARCHOS VR Glasses [14], and Starbreeze StarVR [15]. Most of these upcoming virtual reality products feature at least head orientation tracking and some also position tracking. The goal of head movement tracking is to allow immersion through motion parallax depth cues. [16] One of the major challenges toward this goal is the delay between head movement and display updates.

High delay is bad for the immersion effect. Even if rendering throughput is sufficient to drive the HMD at high frame rates, the delay might be too high. Some of the delay is due to the design of traditional 3D rendering APIs. They utilize frame buffering queues to smooth out frame rate variations.

Even without such buffering the delay between input and display is at least as long as it takes to render one frame. This time period may be too long. John Carmack [17] has proposed "Asynchronous Time Warp" as a solution to reducing the latency. Both Nvidia [18] and AMD [19] have announced support for it. As we shall see later, this kind of latency compensation one existing application of image based rendering techniques.

2.3 Cloud Game Streaming

Lately there has been some interest in streaming live games over LAN or even internet. Image based rendering might have a valid role to play.

2.3.1 Disparity between Display and Network Link Bandwidths

The latest DisplayPort 1.3 standard by VESA specifies maximum 25.92 Gbit/s data rate after overhead [1]. This is barely enough for 4K displays at 120 Hz refresh rate. Even a more common 1080p resolution at 60 Hz requires about 3Gbit/s of bandwidth just for visible pixels in addition to timing information sent over DisplayPort.

Let's compare this to typical network bandwidths. Wired LAN with Gigabit Ethernet can transfer a maximum of 1 Gbit/s before overhead. Tablet Wi-Fi can

manage about 471 Mbit/s real-world performance [20]. Category 9 LTE downlink specifies theoretical maximum speed of 452 Mbit/s [21]. Although such practical speeds have not been yet reached with LTE: Qualcomm is claiming 410 Mbit/s real-world performance with their latest Snapdragon 810 flagship SoC [22]. We can see that these network downlink speeds are about one or two orders of magnitude slower than what display connectors are capable of.

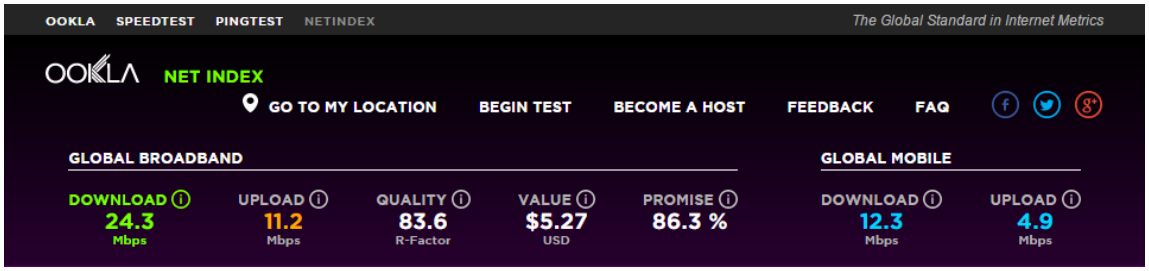

Figure 1: Average global broadband and mobile download rates [23] according to Ookla.

In figure 1 by Ookla we see that global average downlink speeds for broadband and mobile internet connections are 24.3 Mbit/s and 12.3 Mbit/s respectively. Thus it appears that the average network bandwidths are further one order of magnitude down from best available technology. Even for the relatively conservative case of 1080p resolution at 60 Hz, streaming over average broadband needs to compress video more than 100 times.

2.3.2 Interactive Video Streaming with Traditional Video Compression

Efficient compression ratios in excess of 1:100 are possible for common encodings such as H.264/AVC. E.g. a 1080p resolution and 60 Hz frame rate version of the Big Buck Bunny animation clip [24] has been encoded in H.264/AVC at 4 Mbit/s bitrate. This video is compressed 746 times compared to similar uncompressed 24 bit color video.

However unlike non-interactive video streaming, live streaming of interactive content is latency sensitive. Most conventional video compression methods such as H.264/AVC depend on being able to estimate motion by comparing a frame with past and future frames. Future frames are available only in the future, which means compression of frames needs to be delayed until newer frames are available, introducing latency. Alternatively if a traditional compression method only uses past frames in motion estimation, compression ratio may be reduced.

Also with video usually some average bitrate is targeted, but the bitrate of individual frames may vary considerably. Especially H.264/AVC has high variability in bitrate [25]. Buffering is used to hide variability, which may introduce delay. Instead of variable bitrate, variable image quality may be targeted [25]. However in this case lowest acceptable bitrate is bound by the image quality of the least compressible frames.

Practical cloud gaming delay measurements have been made by Chen et al. in 2011 using two services: OnLive and StreamMyGame [26]. They segmented the total delay into three component delays using estimation techniques:

- Network delay: the delay from network transfers to and from the server.
- Processing delay: the delay caused by the server processing user input into a frame.
- Playout delay: the delay between receiving the encoded frame data and presenting it on display.

They reported processing delays between 110 ms and 471 ms. The reported playout delays were between 17 ms and 24 ms. The lowest combined processing and playout delay was 135 ms. They consider the variability of processing delays to be due to choice of hardware used.

2.3.3 Possibilities with Image Based Rendering

Instead of traditional video compression, a cloud gaming system could use image based rendering to reduce bandwidth requirements. In such a scheme high quality reference frames would be transferred with an attached timestamp at some low rate. In addition to color information, each transferred frame would include depth and velocity buffers. Given a transferred reference frame, arbitrary number of future frames may be extrapolated with image based rendering.

There are at least four benefits to this approach compared to traditional video compression:

- Exact velocity information is easy to compute during vertex shading stage of rendering. In comparison motion estimation is a coarse image based approximation technique which may be computationally complex and may depend of information from future frames.
- The reference frame need not be displayed itself, only extrapolated frames. This way the system can compensate for view change latency by reading input locally and for scene change latency by using the velocity buffer.
- Any arbitrary frame rate can be extrapolated locally. A myriad of client display device refresh rates can be supported. Yet in competitive online games no unfair extra information needs to be provided. This also helps to hide small performance variability due to server load variability.
- If the server determines a rendered reference frame to be too similar to previously transferred reference frame, it may decide to skip transferring it to conserve bandwidth.

There may be disadvantages. More information needs to be transferred per pixel in each reference frame. This is somewhat offset by the need to transfer the estimated motion information for predicted frames in traditional video compression schemes. Though if large groups of pixels are moving in unison, velocity information may be spatially very homogeneous. Such a situation is well suited for compression with

run-length encoding. Conceivably depth data could also be compressed with a lossy or a lossless compression algorithm.

While some delays can be hidden, it's not possible to do so in all the cases. E.g. when a player shoots an enemy, there can't be any feedback before the shot is registered on the server and a new reference frame is sent back depicting the result. This is fundamentally the same delay problem that locally rendered multiplayer games face. Ways to mitigate the problem exist [27] and can be perhaps reviewed for usefulness with image based rendering.

3 Human Visual System

The goal of frame rate upconversion is motivated by desire to achieve higher frame rates. The motivation for high frame rates in turn stems from the way human visual system interacts with display devices. To understand why, a brief overview of human visual system is given in this section.

3.1 Eye Integration

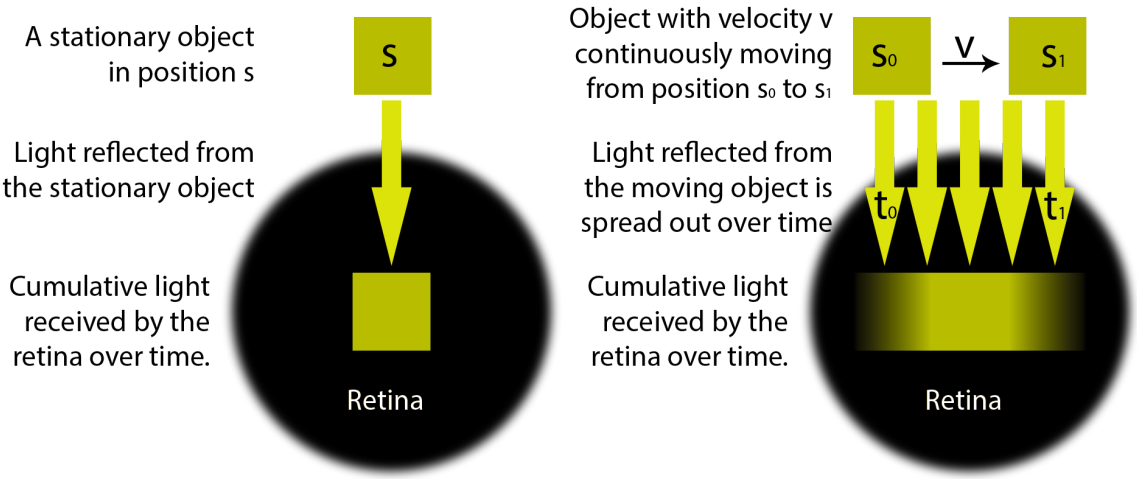

Figure 2: Illustration of image formation on the retina from light reflected by a square object. On the left the object is stationary. On the right the object is in motion. In both cases the eye itself is stationary.

Human vision begins when light coming from some object excites the photosensitive photoreceptor cells on the retina. The process is continuous and light over a period of time is averaged together, or said to be integrated. The effect is equivalent to a temporal low-pass filter [28]. According to Burr [29] the integration period in daylight is about 120 ms. In Westerinks [30] interpretation of Blommaerts work [31], a somewhat lower figure of 50 to 100 ms is quoted.

If the eyes and the world each remain motionless, then the light from the object would be integrated onto constant position on the retina, producing a strong and

sharp signal. This situation is illustrated in figure 2 on the left. If anything in the scene moves, then light coming from it will get spread over a wide area on the retina, resulting in smeared or blurred image and a weak signal. This situation is illustrated in figure 2 on the right.

The model just presented is a simplification. In reality human visual system is more complex. It has sharpening methods that reduce the perceived amount of blurring [29] [32] that would be expected purely based on the integration period. Such details are however beyond the scope of this thesis.

3.2 Smooth Pursuit

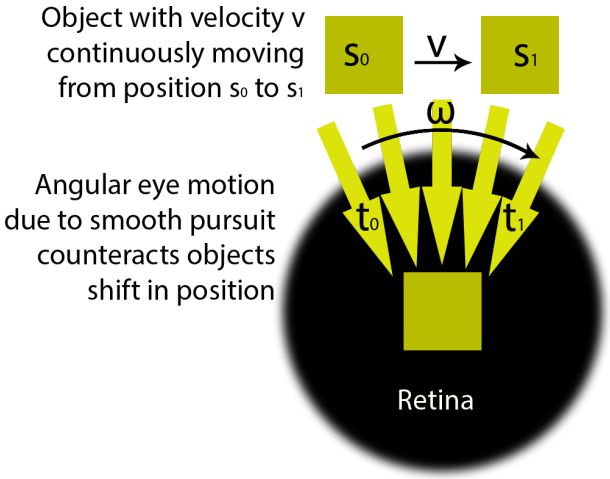

Figure 3: Illustration of image formation on the retina when the eye is in smooth pursuit of the object.

With eye integration, it's possible to view static objects in good detail, but not moving objects. Yet it's often the moving objects that are the most interesting—static objects, such as rocks, tend to be less consequential than moving ones, such as tigers. Human visual system has the ability to translate moving objects into stationary ones on the retina by automatically tracking them [33].

When the eye tracks a moving object, light from it gets projected onto a constant position on the retina producing a sharp image of it. This phenomenon is called smooth pursuit. Figure 3 illustrates how smooth pursuit counteracts object movement.

3.3 Impulse Type Display Image Formation on the Retina

Cathode ray tube (CRT) displays are impulse type displays or sometimes also called stroboscopic displays [35]. Impulse type displays emit light in very short and very intense impulses once per each screen refresh. The CRT emission time is in the order of 50 μs [36]. In figure 4 a CRT display response is shown when driven with a step signal, i.e. source signal is turned on and held on. A CRT display responds by repeatedly emitting light in short impulses once per each screen refresh period $\mathbf{t}$.

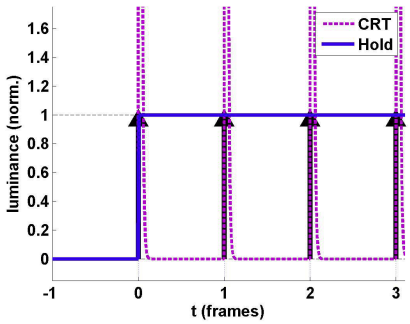

Figure 4: Impulse type (CRT) and ideal hold type step responses [34]

Between the short impulses such displays remain relatively dark for the remainder of the refresh cycle. With e.g. 60 Hz refresh rate, the display would be emitting light only 0.3 % of the frame period. In essence the monitor is flashing at the frequency of the refresh rate. Normally this flashing is not perceived because eye integration causes averaging over the light impulses and dark periods. However with low frequency the flashing becomes visible. The effect is called flicker [34].

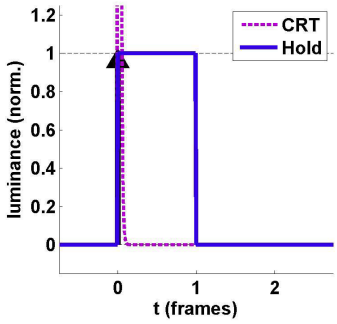

Figure 5: Impulse type (CRT) and ideal hold type impulse responses [34]

Consider what happens when the eye is tracking a moving object presented on an impulse type display. The image is drawn on different spatial locations of the screen, so pixels turn on and off rapidly. In figure 5 illumination responses are shown when display pixels are driven with an impulse signal, i.e. a pixel should emit light for only one refresh cycle. Because the eye is following the object with a smooth motion, whenever during an impulse of light lands on the retina, the light always lands on the same location. Because the light is not spread out, a sharp image is produced on CRT displays [34].

3.4 Hold-Type Display Image Formation on the Retina

Hold-type displays, such as most LCD, deliver light differently than CRT displays. Most LCD displays remain constantly illuminated by the backlight. As an exception some LED backlights in some LCD displays may indeed flash. The technology is

called pulse width modulation [37] and the goal is to dim the display to conserve energy. Such backlights are not considered any further.

Before the light reaches the eye, it travels through the LC cells, one for each subpixel, which filter the light intensity. By varying the driving voltage of LC cells, the amount of light passing through the filter is changed. Changes to LC cell driving voltage occur during screen refreshes, corresponding to impulses on CRT displays. Unlike idealized hold-type displays, real life LC cells have latency and don't change state immediately [34].

Unlike the CRT display which remains dark between the impulses, LCD holds a static unchanging image between individual refreshes. This is where the term hold-type comes from. More precisely such displays are said to have sample & hold characteristics. A moving objects location is sampled at the refresh rate and then held for the remainder of the refresh period. In figure 4 a hold-type display, when driven with a step signal, responds by turning on during the next screen refresh and continuously emitting light afterwards.

There are two relevant consequences to hold-type characteristics. The first consequence, an advantage, is that hold-type displays can't be inflicted with flicker no matter how low the refresh rate [34]. This is because the backlight isn't flashing.

The second consequence is motion blur, a disadvantage. On an impulse type display animation is achieved by translating the location of the object during each impulse. Because light is transmitted only during the short impulses the objects position only needs to be correct during those very short moments. On a hold-type display animation is achieved similarly by changing the objects location during each screen refresh. Between two refreshes however the display shows a static unmoving object, constantly transmitting light, as in figure 5.

Consider what happens when the eye is in smooth pursuit of a moving object, such as the ball in figure 6 by Didyk et al. [38]. The eye is following the object in a smooth continuous fashion like the empty circle in the figure. In reality the ball would also move continuously as in the top row. A hold-type display however only updates the image during each refresh cycle, like in the middle row. From the retina frame of reference, the object seems to be continuously falling behind until it skips forward again during the next screen refresh. This backwards motion can't be perceived when the eye is integrating light over a period of time longer than the frame period. Instead the object appears blurred along its motion trajectory because the light arriving during each frame period is not focused on a fixed area of the retina. This is shown in the bottom row of figure 6.

The width of the apparent blurring is equal to the length that the moving object falls behind before the screen gets updated again. This length in turn depends on the objects speed and the time between screen refreshes, i.e. the hold period. The amount of motion blur and the refresh period are in linear relation. [28]

An additional source of blurring comes from the LC latency, but it's distinct from the motion blur caused by hold-type characteristics. Its contribution can be effectively mitigated with e.g. LC cell voltage overdrive [39]. As such, LC latency is not considered further from this point forward. Instead ideal hold-type characteristics are assumed for LCD panels.

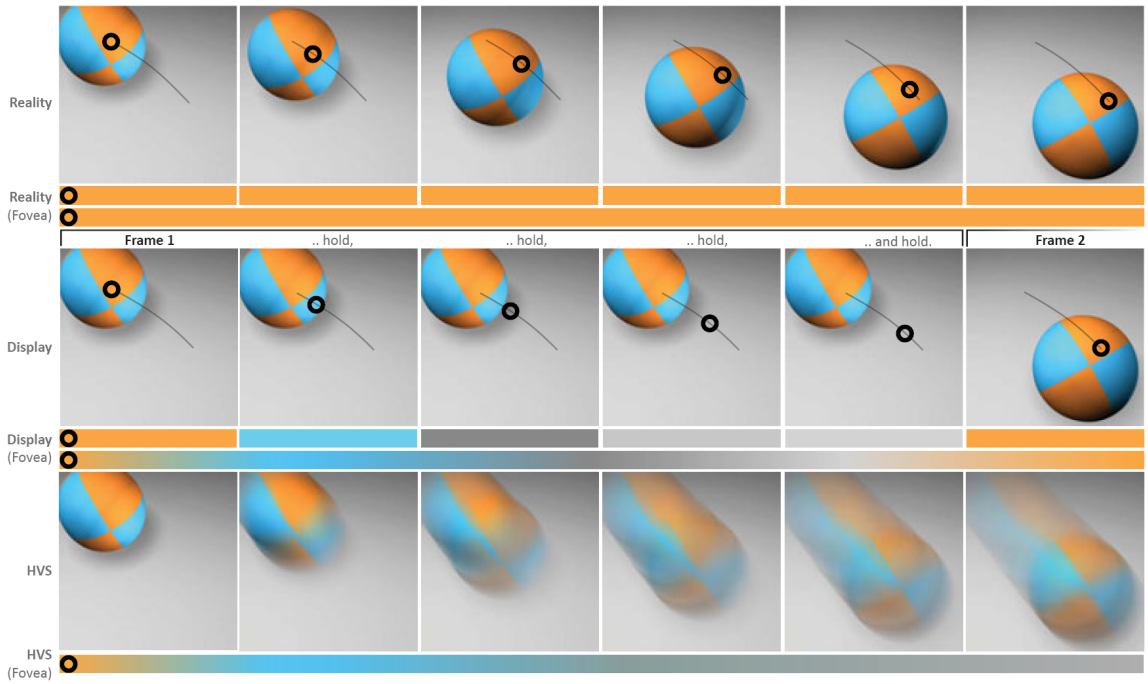

Figure 6: Hold-type motion blur as visualized by Didyk et al. [38]. The empty circle is the eye fixation point.

3.5 Motion Blur Reduction

A number of approaches have been investigates for motion blur mitigation in LCD displays, e.g. work by Klompenhouwer [28]. One approach involves strobing the backlight to emulate impulse type light delivery. The goal is to reduce the hold period to a shorter pulse, like with impulse type displays. Doing so requires a special backlight that can be flashed quickly in sync with the refresh rate. As a side effect, brightness is reduced because the backlight would be turned off for a large portion of the refresh period. Note that while strobing backlight technology is similar in implementation to pulse width modulation technology [37], it has a very different goal. Pulse width modulation is applied to intentionally dim a LED backlight, ideally by switching on and off at a very high frequency.

Another approach, called black frame insertion, involves inserting black frames between animation frames. Doing so has the effect of strobing the LC cells instead of the backlight. Again, brightness is reduced because of the alternating black frames. Further, inserting black frames reduces the effective frame rate available for animation. Also flicker may be introduced at too low refresh rates just as with impulse type displays, which this technique is emulating.

Finally, instead of simulating impulse type displays with black frame insertion, one may simply insert actual real frames, i.e. raising the frame rate. In comparisons with strobing backlight and black frame insertion this method is called high frame rate (HFR) driving. High frame rate reduces motion blur by simply reducing the sample & hold period. HFR doesn't reduce display brightness nor does it reduce usable refresh rate. Instead HFR is limited by source material availability.

What is considered HFR varies by context. In cinema, where the ubiquitous 24 Hz frame rate dominates, 48 Hz is considered HRF as introduced by The Hobbit in 2012 [40]. Television material is commonly available only in 25/50 Hz (PAL), and 30/60 Hz (NTSC). Televisions that support higher refresh rates get around source material limitations by using frame rate upconversion with motion estimation based frame interpolation, called motion compensation. Such algorithms have been around already before the LCD TV era, e.g. as described in [41]. Essentially motion estimation algorithms detect patterns of motion in between source frames and based on that information compute new intermediary frames.

In the context of PC gaming LCD displays, anything higher than the long standing LCD industry standard of 60 Hz may be considered high, e.g. 120 Hz and 144 Hz are common gaming display refresh rates. Unlike pregenerated animation material, in live computer animation all the frames are generated locally on demand and source material availability is not strictly the limiting factor. Instead the speed of the GPU becomes the limiting factor. The frame rate which any given GPU can produce is in turn limited by the complexity of the scene geometry, among other factors. The amount of motion blur produced is thus in inverse relationship with hardware computation power and scene geometry complexity.

4 Theoretical Background

To fulfill the first goal set in the introduction section, a literary study was performed. A short overview of a traditional rasterizer based rendering pipeline is given first though to establish some basic concepts.

4.1 Geometry Based Rendering Recap

Frame rate upconversion can't be discussed without first reviewing some basic concepts of traditional geometry based rendering. Only details which are absolutely necessary for later discussion are reviewed. For a more in-depth overview of a typical rendering pipeline in general the reader is referred to abundant literature on the subject, e.g. [42] or [43].

4.1.1 Viewing and Projection Transformations

Let's consider the mathematic relationship between geometric and pixel based representations of a 3D scene. In most cases of 3D computer graphics the scene objects, or rather their surfaces, are encoded as a collection of connected triangles. These triangles need to be projected onto a 2D display surface.

To produce a 2D image of a 3D scene, a viewpoint and gaze direction must be chosen. A viewing transformation performs a change of coordinates to the triangles in the scene so that the viewpoint becomes the origin and the gaze direction is aligned to some axis, often the negative z-axis. Projection transformation further projects the triangles onto a plane representing the flat display area.

Following, only a high level review of the transformations is given. The various used transformations are explained in more detail by e.g. Lengyel [43] in, chapter 4: Transforms and chapter 5: Geometry for 3D Engine.

Scene geometry may initially be loaded from various files and formats with each having its own coordinate system, referred to as model coordinate system. Eventually when they are all placed in the scene they will be transformed into a common coordinate system. This coordinates system is called the world space, a three dimensional Euclidean coordinate system. For simplicity, from now on all scene geometry is assumed to be initially in world coordinates.

Commonly the transformations are represented with matrices. Transformations are performed by multiplying the object coordinates with the appropriate matrix. Commonly in computer graphics, three dimensional vector coordinates are represented as four dimensional homogeneous coordinates, where the fourth parameter, called w, signifies scale. For more information on homogeneous coordinates, see for example [43] chapter 4.4 on Homogeneous Coordinates. As with vectors, matrices too are in four dimensional homogeneous representation.

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w = 1 \end{bmatrix} = \mathbf{v_w} \tag{1}$$

For example in equation 1 an arbitrary point belonging to the scene geometry is represented in world space coordinates with a homogeneous vector $\mathbf{v_w}$ composed of the three spatial coordinates x_w, y_w, z_w and by the additional homogeneous scaling value w_w, which is commonly initialized to exactly 1 while dealing with world space.

$$\mathbf{v_v} = M_{w2v} \times \mathbf{v_w} \tag{2}$$

Next in equation 2 a viewing transformation is performed—essentially a change of coordinate base. A vector in world coordinates is multiplied with an affine transformation matrix M_{w2v}, usually a composite of translation and rotation transformations. The change of base centers the scene origin at the chosen viewpoint and rotates all the objects around the origin to match the desired gaze direction. As a result, the geometry is said to be transformed into view space, sometimes also called eye space or camera space.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \mathbf{v_c} = M_{v2c} \times \mathbf{v_v} \tag{3}$$

Next in equation 3 a planar projection transformation is performed with the matrix M_{v2c}. As a result the geometry is said to be transformed into clip space, represented by the vector $\mathbf{v_c}$. Details of why this stage is called the clip space are omitted here. Suffice it to say, geometry outside the viewable volume of space is easily detected and clipped at this stage.

$$\begin{bmatrix} x \\ y \\ Z \\ 1 \end{bmatrix} = \mathbf{v_d} = \mathbf{v_c}/w_c = \begin{bmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \\ 1 \end{bmatrix} \tag{4}$$

Finally in equation 4 clip space coordinates are normalized by dividing with w_c. This step is sometimes called perspective divide. After this step the coordinates are said to be in window space or normalized device coordinates, depending on source. The w scale value is reduced to exactly 1, and can be ignored from now on. In this space all the visible object x and y coordinates should lie between -1 and 1, corresponding to horizontal and vertical screen edges. The Z coordinate retains the depth information, encoding the original distance of each point of the geometry from the screen projection plane. While the coordinates correspond to the display device edges, at this stage the geometry is still just geometry, not color samples nor pixels. Also there is no notion of display resolution yet. That comes after rasterization.

4.1.2 Rasterization

To view the scene on a display, the projected 2D geometry needs to be converted into an array of pixels. Most modern GPU hardware converts geometry into pixels with a process called rasterization. There are alternatives to rasterization, such as ray casting, but these methods are not considered. Also rasterization may be implemented in multiple ways. An overview of relevant parts is given.

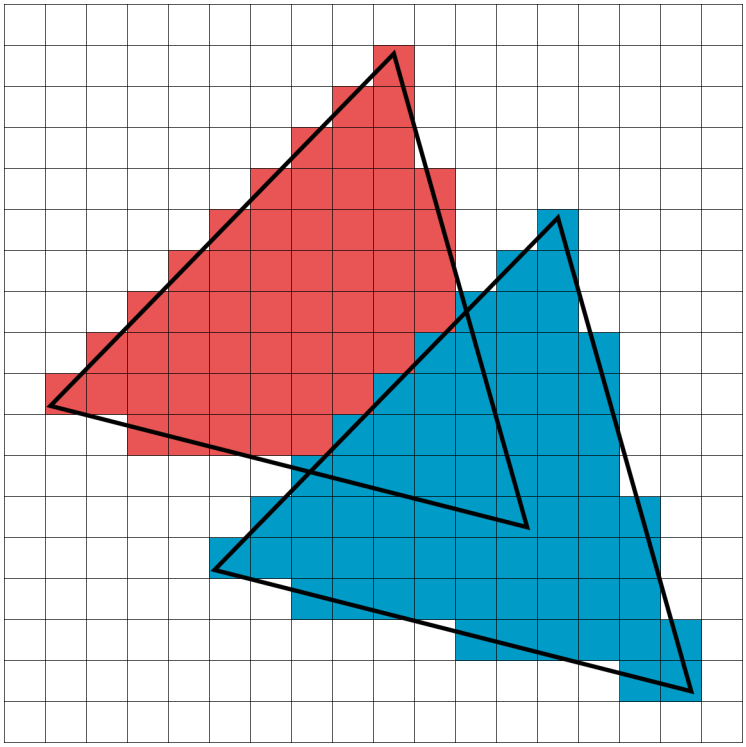

Figure 7: Two rasterized triangles. The red triangle is occluded by the blue one.

Regardless of implementation details, the goal of rasterization is to convert a set of 2D triangles into a set of color fragments, which represent samples taken from the surfaces of the triangles at even intervals. Color fragments roughly corresponds to the notion of pixels. Each fragment may become a pixel in the output image, but this is not a given. Essentially rasterization determines which ones of the screen pixels a triangle covers. In figure 7 two rasterized triangles are shown. The grid represents pixel spacing of the display device.

The fragments are colored according to the color of the triangle surface point which it represents. The exact method of determining the color of a fragment is a process called shading, but is not considered here in more detail. E.g. Phong shading [44] is a commonly used algorithm. In figure 7 the triangles are colored with a single color.

After rasterization and shading the color fragments should be saved to the color buffer, except the ones which are occluded in the view. Since the triangles originally come from a 3D space which was flattened onto a projection plane, many of them likely overlap. Only the rasterized fragments corresponding to the parts of triangle surfaces which are in front of other surfaces should be places in the color buffer.

A method of depth sorting overlapping fragments is thus needed. Only one method is considered here: Z-buffering. From equation 4 the Z-values of the triangle corners are interpolated for each rasterized fragment. Also another buffer in addition to color buffer is used, called a Z-buffer. A fragment and its Z-value are stored in the color buffer and Z-buffer only if it is closer to the projection plane than the previous value in the same position; otherwise it's dropped. In the end, the color buffers will contain colors samples only from the parts of surfaces which weren't occluded by others. The front most color fragments have become pixels in the color buffer. In figure 7 the blue triangle is closer to the viewpoint than the red triangle. Consequently color from some of the red fragments is not saved as pixels into the color buffer.

There are more steps in a typical rasterization based rendering pipeline than presented here, however they are not relevant from the point of view of this thesis. For example Shirley et al. explain rasterization in [42], chapter 3.6: Triangle Rasterization.

4.2 Image Based Rendering

A powerful GPU can help drive a high frame rate necessary for reducing motion blur. Similarly reducing scene geometry complexity can help raise the frame rate by reducing the vertex processing load [45]. Both are straightforward brute force solutions. Another approach, called image based rendering, is considered here for rendering some fraction of all the displayed frames instead of only using conventional geometry based rendering.

Figure 8 illustrates the basic idea. In image based rendering, the 3D scene is represented by a collection of snapshot reference images of the scene [46]. From these reference images, new viewpoints of the same scene are produced. To do so, the reference images need to be paired with depth information. In some cases depth

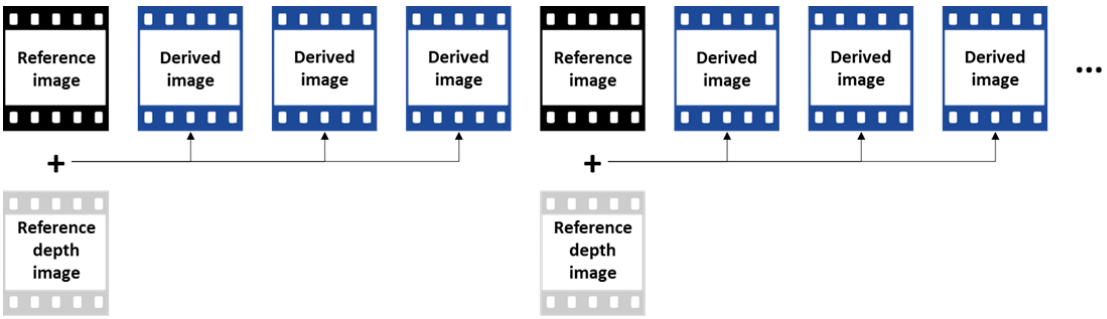

Figure 8: A sequence of reference images and derived images.

information may be inferred from the reference images themselves [47] [48] or it may be provided in form of an explicit depth image. However under either condition, there is no explicit geometric representation of the objects the scene is composed of.

The motivation for using image based rendering stems from desire to decouple frame rate performance from geometric complexity. Indeed, McMillan [47] suggests that image based rendering might be more efficient than traditional geometry based rendering once the amount of geometry surpasses the number of rendered pixels. The reference images may be photographs or computer generated images, though from now on only computer generated images are consider. Also an explicit depth image (Z-buffer) is assumed to be available.

4.2.1 Frame-To-Frame Coherence

In frame sequence animation, temporal sampling rate, i.e. frame rate, is inversely related to the amount of spatial change in consecutive frames. Depending on source, the frames are said to have temporal coherence or frame-to-frame coherence.

The possibility of taking advantage of frame-to-frame coherence in context of 3D computer animation has been identified at least as early as 1981 by Hubschman [49] in his Master's thesis. The early focus is on exploiting frame-to-frame coherence for optimizing hidden surface computations [50].

The idea of using temporal coherence for reusing the computed color values seems to have come later. In 1988 Badt [51] presents two algorithms for exploiting temporal coherence with ray tracing. The first one is used for assessing temporal coherence between an old frame and a yet unrendered frame. They test a number of randomly distributed pixels by casting rays for those pixels. Using their algorithm they determine coherence in the test locations. If coherence is low, rays are cast also for the surrounding pixels in flood fill manner while coherence remains low. The other algorithm is used for reprojecting the pixels of the remaining highly coherent image areas from previous frame to the new frame. This is an approximation method with error proportional to desired coherency threshold. Reprojection is said to be less expensive than casting new rays for all the pixels.

Similar work on ray tracing has been carried out by Chapman et al. [52]. Also other ray tracing methods attempt to utilize temporal coherence while achieving exact result [53] instead of approximations. Also 3D cinema could use image-based

rendering to implement the motion parallax depth cue [16].

This early work deals with offline animation rendering with ray tracing. Systems more suitable for live rendering with rasterizing are considered later by McMillan, Bishop, and Mark [54] [55] [56]. High frame rates correspond to short frame periods, which in turn should lead to higher possibility of temporal coherence. While high frame rate is needed to reduce motion blur, rising temporal coherence seems to imply that consecutive frames don't actually contain all that much new information. In 3D computer graphics spatial change corresponds to the amount the visible surfaces change position, or rather the pixels representing samples of those surfaces. Indeed it's rather likely that most surface point visible in one frame are still visible in the next frame and positioned somewhere close by.

4.2.2 Reprojection

Figure 9: Original image of a teapot. Image captured from video by username "eVRydayVR" on Youtube [57] at time 11:18.

Reprojection is the first step in image based rendering. Consider any pixel, or color fragment if you will, at some x and y coordinate of a rasterized image. That pixel represents a color sample taken from a single point of a surface of the original scene geometry. If the image was instead rendered from another viewpoint and gaze direction, then what would be the x' and y' coordinates which this surface point would have been rasterized onto? The process of reprojecting the pixel deals with just that question. Figures 9 and 10 show a teapot before and after all the pixels have been reprojected to a new viewpoint.

As a stopgap, a coordinate system is needed which is independent of both the old and new viewpoint. The world coordinate system originally used to describe

Figure 10: Image of a teapot after pixels have been reprojected to a new viewpoint. Image captured from video by username "eVRydayVR" on Youtube [57] at time 11:30.

the geometry in equation 1 is such a coordinate system. To get a representation of the pixels in world coordinates, the projection transformation and the view transformations need to be performed in reverse for each pixel in turn. To do that inverse matrices are calculated from the known projection and view transformation matrices that were originally used to produce the image.

Before the geometry was rasterized the geometry was represented in normalized display coordinates (equation 4). When dealing with rasterized images, the spatial coordinates are usually given in resolution dependent display coordinates and need to be normalized first. The x and y coordinates are commonly in a range between zero and the pixel count of the respective axis. E.g. $0 \ldots 639$ and $0 \ldots 479$ for VGA resolution. When normalizing, it's important to note that while scene geometry is defined in terms of singular points in space, pixels are two dimensional rectangular areas. On the other hand the colors of all the pixels were sampled from some singular points on the surfaces of geometric objects in the scene. Because of this pixels can be treated as points and one needs to consider the point within the pixel boundary where the sample was originally taken. The exact location varies per rasterizer implementation. If for example the sample was taken from the center of the pixel, 0.5 needs to be added to the unnormalized pixel coordinate before normalization.

$$\begin{bmatrix} x \\ y \\ Z \\ 1 \end{bmatrix} = \mathbf{v_d} \tag{5}$$

For the purpose of discussion it's assumed from now on that the x and y coordinates are normalized to the range between -1 and 1. In equation 5, each pixel coordinate of the image is represented with a normalized display vector, just like each geometry point was represented before being rasterized. The Z value is read from the Z-buffer.

A pair of homogeneous vectors are considered equivalent if they are scalar multiples of each other [58]. Remember, normalized display coordinates were originally produced by dividing clip space coordinates with w. Consequently clip space coordinates and normalized display coordinates are scalar multiples of each other and thus homogenously equivalent. Thus when performing transformations with homogeneous coordinates, normalized display coordinates can be substituted for clip space coordinates.

$$\mathbf{v_v} = M_{v2c}^{-1} \times \mathbf{v_d} \tag{6}$$

$$\mathbf{v_w} = M_{w2v}^{-1} \times \mathbf{v_v} \tag{7}$$

Moving on, in equations 6 and 7 the vector $\mathbf{v_d}$ is multiplied by both the inverse of the projection matrix M_{v2c} and the inverse of the view matrix M_{w2v} to yield the world coordinates $\mathbf{v_w}$ for the pixel.

$$\mathbf{v'_c} = M_{v2c} \times M'_{w2v} \times \mathbf{v_w} \tag{8}$$

Next, in equation 8 the world coordinates are transformed back to clip space. This time a new view transformation matrix $M^{'}_{w2v}$ is used to get a new viewpoint and gaze direction. The same old projection matrix M_{v2c} can be used, but of course another one may be substituted if different projection is desired.

$$\begin{bmatrix} x' \\ y' \\ Z' \\ 1 \end{bmatrix} = \mathbf{v'_d} = \mathbf{v'_c}/w'_c = \begin{bmatrix} x'_c/w'_c \\ y'_c/w'_c \\ z'_c/w'_c \\ 1 \end{bmatrix} \tag{9}$$

Finally in equation 9 normalizing with $w^{'}_c$, yields reprojected normalized display coordinates. It's possible for some of the pixels to lie outside of the display area when viewed from this viewpoint. In that case the coordinates will lie the normalized -1...1 range.

4.3 Reprojected Image Reconstruction with 3D Warping

Reversing the projection and viewing transformations for each pixel in a rendered image, as done in equations 6 and 7, essentially yields a cloud of color samples floating in world space where the visible surfaces of the original geometry were. Such floating pixels are visible in figure 10 in places where the cloud is sparse. Clearly a better method of image reconstruction is needed than just translating the pixels to new reprojected positions.

While the viewing and projection transformations were reversible, the rasterization process itself cannot be reversed in any trivial way. Human observer may intuitively

see geometric structures if viewing such a cloud, but any explicit link to the original geometric model data is lost. That being said, plenty of prior work investigates possibilities of exploiting the available color and spatial information despite not knowing the underlying geometry.

In 1995 McMillan and Bishop [54] present a head tracking virtual reality system that uses a method called image warping for synthesizing a wide range of viewing positions from a set of reference images. In 1996 Mark, McMillan and Bishop [55] follow up with a system for interactive rendering server with latency compensation. Also in 1997 they [56] describe a system for upconverting a rendering systems frame rate. While the goals of the three publications are different they are all using a 3D image warping technique to produce new viewpoints from one or many existing viewpoints of a 3D scene.

4.3.1 Image Warping

The basic idea of image warping is described in a book by Wolberg [59]. Wolberg offers the following analogy to help visualize the process:

> Imagine printing an image onto a sheet of rubber. Depending on what forms are applied to that sheet, the image may simply appear rotated or scaled, or it may appear wildly distorted, corresponding to the popular notion of a warp.

Warping maps each integer pixel coordinate x and y in the original image to a new spatial coordinate x' and y'. One could simply round the x' and y' coordinates to nearest integer pixel coordinate values and place them in the output image. Doing so would however results in gaps where distances between pixels became stretched (as in figure 10).

Conversely, should an area of the image become compressed, some pixels would map on top of each other in arbitrary order. The resulting image would look like a variable density cloud of floating pixels instead of a solid rubber sheet. Furthermore the resulting image may well be rendered in higher resolution than the source image, resulting in considerable stretched out pixel spacing. Clearly warped image reconstruction is a nontrivial problem to be solved.

In addition to the aforementioned naive technique of simply moving the source pixel to a new place in the output image, at least two reconstruction methods have been considered. On method, called splatting, was used for example in [55] and [56]. Another possibility is to use a mesh based approach, such as in [56].

4.3.2 Splatting Based Warp Reconstruction

Splatting is a simple process of reconstructing a new viewpoint. Early on splatting has been investigated for rendering of 3D voxel volumes [60]. Essentially each sample in a voxel volume is represented by some color intensity kernel which is projected to the screen. E.g. a kernel may be sphere, which is projected to the screen. [61] The intensity contribution of a projected kernel may vary with e.g. a Gaussian function.

In [55] splatting is described as a reconstruction technique which draws the original pixels in their reprojected positions as color splats of varying size. They don't describe the exact shape of the splat kernel that they used, but they mention that sizing is very important to prevent gaps between splats. The size of splat depends on its relative distance from the original viewpoint and the new viewpoint. Z-buffering was used to preserve correct depth ordering. In [56] the same splatting technique was also used for their real-time implementation of a frame rate upconversion system, while using mesh based approach for their offline implementation. They note that splatting based reconstruction is inferior in image quality to a mesh based approach. They report rough edges on under-sampled surfaces and occasional pinholes.

After viewpoint change new surfaces may become disoccluded. Disocclusion is a fundamental problem with image-based rendering. Splatting offers no approximation for such image areas and gaps will remain visible in the result. Frame-to-frame coherence however implies, that over short time periods, the problem is not severe.

4.3.3 Mesh Based Warp Reconstruction

An alternative to splatting is a meshed based reconstruction approach. Each pixel in the reference is represented as a node in the mesh. The nodes are connected to the neighboring nodes vertically and horizontally forming a regular grid, so the node x and y coordinates correspond to pixel coordinates. The color and Z-values from the reference images are attached to the nodes.

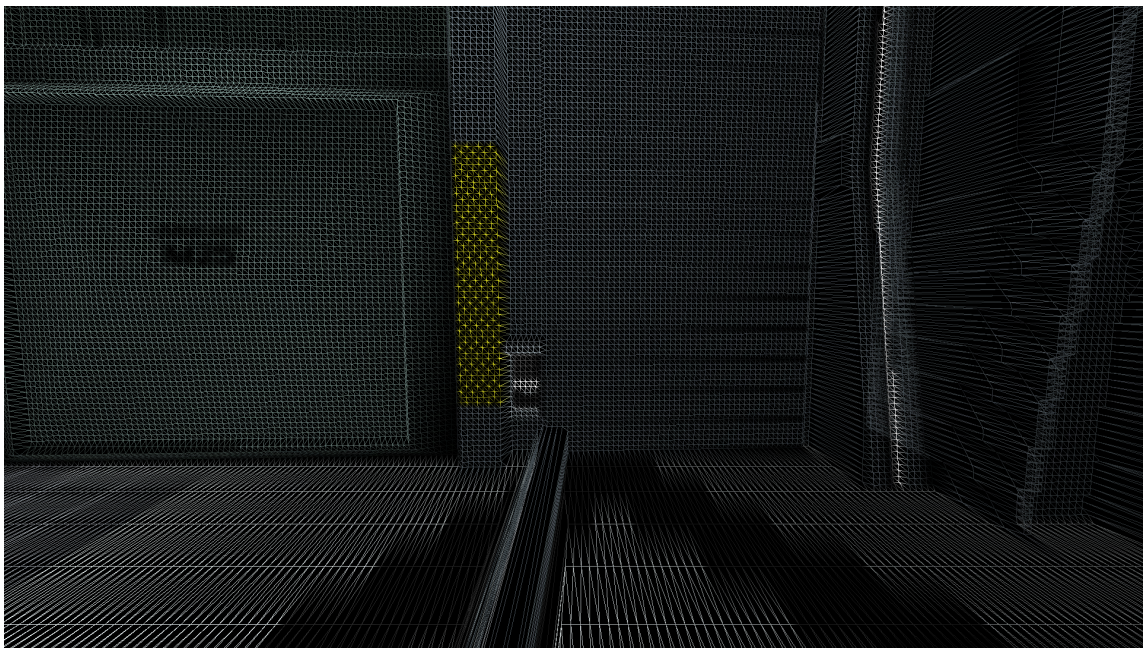

Figure 11: A wireframe representation of a close up view of a warping mesh.

With the addition of the Z-value, each node can be reverse-projected back to 3D world space. A wireframe representation of a close up view of a warping mesh is shown in figure 11 after each node has been reprojected. In world space the mesh

looks like a plastic wrapper enveloped over the original scene geometry from the direction of original viewpoint. Intuitively this analogy is much more reminiscent of the concept of a rubber sheet which Wolberg used to describe warping [59] than splatting. This kind of system is mentioned for example in [54] and [56].

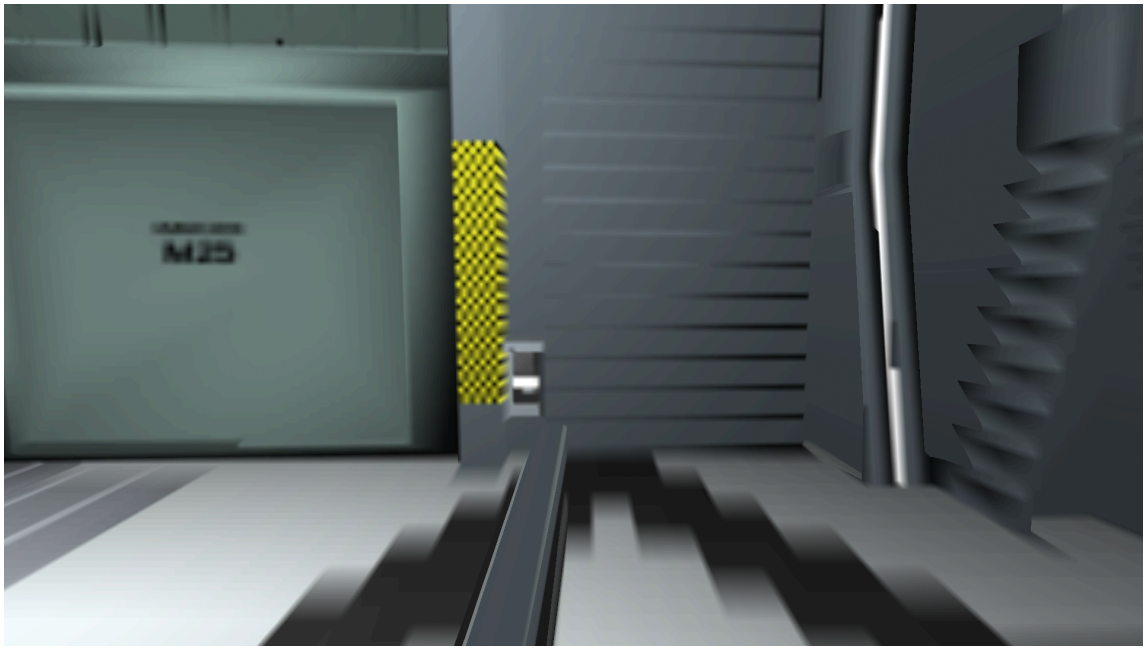

Figure 12: A reconstructed warp.

Color of each node on the mesh is continuously interpolated. For example, such interpolation function could take the form of nearest neighbor interpolation, bilinear interpolation, or bicubic interpolation [59]. In figure 12 the mesh in figure 11 is reconstructed with bilinear interpolation.

Image quality is the main motivation to using a mesh based reconstruction approach. Because the reconstruction is continuous, the will be no gaps between nodes: the primary problem with splatting. Note the bottom area of figure 11. Surfaces which were originally almost perpendicular to gaze direction are clearly undersampled in this close up view. Mesh based reconstruction ensures that there are no gaps like in figure 10. Further, frame-to-frame coherence implies that this kind of exaggerated undersampling can be avoided in most situations.

At this point the mesh is just a geometric entity representing a rubber sheet. Finally after each node is reprojected back to screen space the mesh still needs to be converted into a raster image. Rendering such a mesh is computationally more complex than splatting. Each quad of four nodes forms a polygon surface which needs to be rasterized onto the output image. In total there are as many quads to rasterize as there are pixels in the source image. Each quad may be divided into two triangles, so twice as many triangles need to be rasterized.

Rasterizing the mesh can be done with the same methods that is used normally to rasterize scene geometry . For example the image in figure 11 was rasterized with OpenGL. Consequently the mesh based approach is performance wise limited by

the same rasterization process as normal geometry based rendering. Unlike normal geometry rendering though, the mesh color values come directly from the mesh nodes and no complex shading algorithm is needed. Furthermore the mesh triangle count is fixed to a constant value depending on the source image resolution. While the source scene may contain arbitrarily complex geometry, new viewpoints may be reconstructed with constant effort.

Like with any 3D rendering, with 3D warping a portion of the warped surface may become occluded by another portion when viewed from another location. McMillan et al. use a standard Z-buffering algorithm as part of the resampling process to solve the occlusion problem. While they note that Z-buffering is problematic due to computation power and memory limitations of the time (1995), Z-buffering isn't a concern for modern hardware.

4.3.4 Sparse Reconstruction Mesh

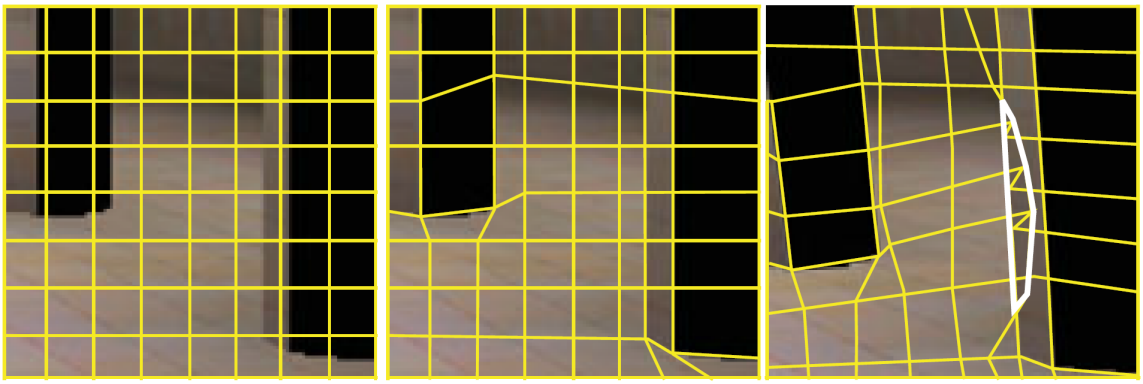

Figure 13: Sparse reconstruction mesh as visualized by Didyk et al. [38]. On the left is the original sparse mesh. In middle image the mesh has been snapped to depth discontinuities. The right image shows the mesh after reprojection.

In Mark et al. [56] and Didyk et al. [38] use an approximation to reduce the node count of the reconstruction mesh. In figure 13 on the left, more than one reference pixels are covered by the grid cells.

It's possible to use a sparser warp mesh where the conceptual 3D rubber sheet formed by the mesh is locally roughly planar. In such a case the skipped nodes have x, y, and Z values that fall roughly linearly and continuously between the retained nodes. In such areas the values in the Z-buffer form a smooth gradient without discontinuities.

If a discontinuity would be present between two sparsely spaced nodes, the skipped nodes would not be properly approximated. Luckily, the nodes forming the grid over the source image need not be regularly spaced. In areas where there are discontinuities in the Z-buffer, the sparse nodes can be shifted so that the mesh is aligned with the discontinuities. That way the areas within a grid cell will represent flat portions that can be interpolated correctly. In middle image of figure 13, the grid has been snapped to depth discontinuities. On the right is the grid after reprojection.

Multiple discontinuities inside grid cells would not be solved this way though. In such a case the grid may be too sparse.

With this optimization each remaining node is not given a color value of the corresponding pixel, but rather its original position x and y values are saved as texture coordinates to the original image. Unlike the node position values, which are reprojected, the texture coordinate are only interpolated for all the mesh surface points in the same fashion as color values would have been interpolated. After rasterization, a texture lookup is performed from the source image to determine target pixel color.

4.3.5 Disocclusion Surface Correction

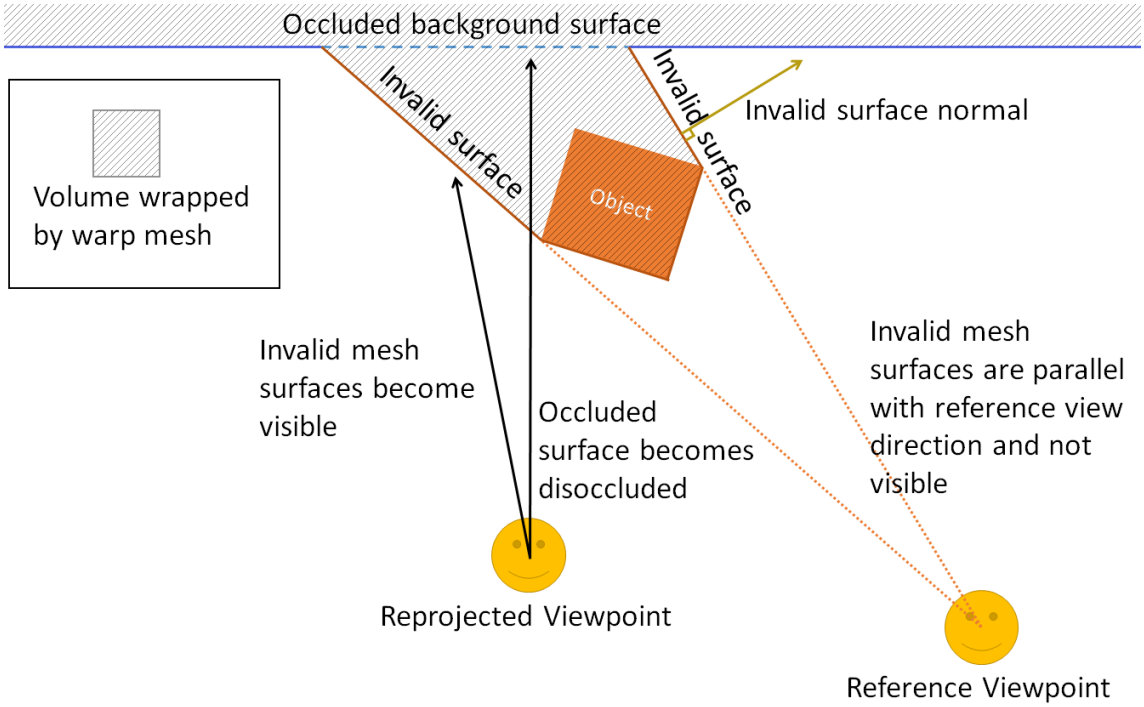

Figure 14: A simple scene where a cube object occludes the background surface.

Consider again the 3D warping analogy where a plastic wrapper is enveloped over the original scene geometry from the direction of original viewpoint. Such a situation is shown in figure 14. All the originally visible surfaces get covered with the wrapper, forming an enclosed volume (dashed area in figure 14). However empty volume behind a foreground object and the background would get also wrapped, or walled off, inside the wrapper mesh. This kind of a formed wrapper surface between the silhouette borders of foreground objects and the background is new to the scene. This invalid surface is not based on the original scene geometry.

In fact this kind of surfaces are artifacts of occluded space, which comes into view after changing the viewpoint and causing portions of the view to become disoccluded. Such invalid surfaces are parallel to a ray cast from the reference viewpoint to the surface (orange dashed line in figure 14). This fact makes them easy to detect. Once

detected there are many ways to deal with them. In figure 14, ideally the invalid surface would be removed and the previously occluded background surface would be shown instead. However with no direct color information available for the newly disoccluded area, some guesswork is needed.

An easy approach would be to color the surfaces by interpolating their colors from the neighboring nodes like any other surface. Doing so would color the surface with a gradient between the background and the surface area. Any surface behind the foreground object is unlikely to be colored similarly to the foreground object, even partly, so this solution is far from optimal.

Another approach would be to skip rasterizing such surfaces altogether and leave the area uncolored. This would leave the foreground part of the mesh disconnected from the rest of the mesh. There would be a black hole visible in the portion of scene background that is no longer occluded by the foreground object. While not incorrectly influenced by the foreground object color anymore, this way also the background influence is lost. In fact it is quite likely that the color of the disoccluded surface would have been similar to the rest of the background. So this is not optimal either.

If the background nodes are taken to represent the best approximation for the disoccluded area, then it may make sense to drop the invalid surfaces as before, disconnecting the mesh. Unlike before, afterwards the mesh would be stitched back together to allow interpolating the background. Unfortunately such stitching is not trivial to do especially in real-time systems. In [56], Mark et al. a simpler method is used. The invalid surfaces are not dropped, but their color is interpolated only from those nodes which belong to the background.

4.4 Latency Compensation with Image Based Rendering

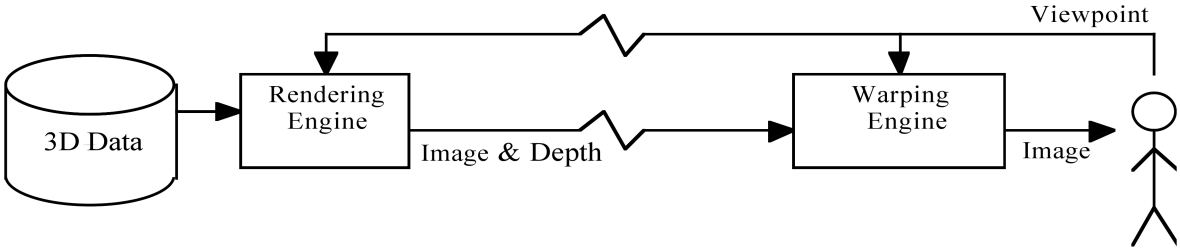

Figure 15: Latency compensating remote rendering system by Mark, McMillan and Bishop [55]

Instead of using image based rendering (only) for frame rate upconversion, it can also be used for latency compensation. In 1996 Mark, McMillan and Bishop [55] presented a system for interactive rendering server with latency compensation. An overview of their system is presented in figure 15 [55].

Viewpoint and gaze direction is controlled by user with e.g. mouse, joystick, head tracking or gaze tracking. The viewpoint and gaze information is sent to the server for rendering. Based on 3D data stored in a database the server computes a new

image from the requested viewpoint. The image is then transmitted back to the user along with color and depth information. The image is not directly displayed to the user yet. Instead, viewpoint and gaze direction is updated from the controller and used to warp the image to match the updated viewpoint. Thus the transmitted image is compensated for latency before displaying it.

They identify two sources of latency in their system. First kind of latency is between the change in controller input and the displayed image. They call this view transform latency.

The second type of latency they identify is between a change in the scene itself and the displayed image. They call this scene change latency. From figure 15 we can see that view transform latency incorporates both delay from uploading the controller input to server as well as downloading the resulting image from the server. Scene change latency has only downlink delay.

While they go on to discuss mitigation of view transform latency with warping in this and future papers [56], at the same time they adopt a fatalistic view of scene change latency:

> Unless we can accurately predict future scene changes, there is nothing we can do about scene change latency. [55]

While their statement is strictly correct, it's certainly not impossible to make reasonably accurate predictions about near future. For example a physics based scene which contains only objects moving at constant velocity is fully predictable using Newton's first law of motion:

> Every body perseveres in its state of rest, or of uniform motion in a right line, unless it is compelled to change that state by forces impressed thereon. [62]

$$x = x_0 + v_{0x}t + \frac{1}{2}a_x t^2 \tag{10}$$

Many dynamic scenes in computer graphics simulate basic Newtonian physics with numeric integration. In such simulation forces, and accelerations, are recalculated only during each simulation step. Between simulation steps, acceleration is constant. Equation 10 [63] formulates such kinetic motion. In the equation x is the new position after time t, x_0 is the initial position, v_{0x} is the initial velocity, and a_x is constant acceleration.

There are a few common cases when objects in a computer generated scene don't behave according to Newtonian physics. E.g. when an object suddenly appears in the scene by teleportation without moving there, we observe a discontinuity in position. Also whenever velocity changes unexpectedly—e.g. a game character comes to a complete stop suddenly as a result of some scripted event—velocity is discontinuous. While such special cases are not realistic, they are commonly used simplifications in computer generated scenes. For the bulk of rendered frames though such special circumstances don't apply.

However when they do appear they cannot be compensated for as previously stated by Mark et al. [55]. They however point out that fortunately scene change latency is not perceivable unless user is interacting with the scene. Without interaction the user simply perceives the scene playing out like a movie. They also think that "scene change latencies of a significant fraction of a second" may in some cases be tolerable even in interactive applications.

$$x = x_0 + v_{0x}t \tag{11}$$

Physics may be simulated at a higher frequency than the frame rate. Acceleration can't therefore be assumed to be constant over the whole frame period. Any predictions of future positions of object must work without explicitly factoring in acceleration.

Equation 11 is the first order Taylor approximation of equation 10 taken near *t=0*. Acceleration is literally taken out of the equation. This makes sense because as the time interval becomes smaller and smaller, the effect of initial velocity is reduced linearly whereas the effect of acceleration is reduced polynomially. But the approximation is valid only as long the frame period is sufficiently short.

How short a time interval is short enough? In a 2010 user study by Didyk et al. [38] they upsample frame rate from 40 Hz source signal to 120 Hz using a similar warping based approach. I.e. a source frame was used to generate 2 extrapolated frames 1/120 seconds apart each. Thus the time difference between the source frame and the 2nd extrapolated frame was 2/120 seconds. They tested multiple scenes of which at least the "fan" scene includes a rotating object affected by normal acceleration. Overall 82 % of test subjects did not report seeing any visual artifacts in a side-by-side comparison with native 120 Hz signal.

5 Task Performance Experiment

In previous sections it has been established that hold-type displays are inflicted with motion blur when depicting moving objects. It has been established that high frame rate mitigates both motion blur and input latency. Finally, image based rendering techniques with 3D warping have been investigated as a method of frame-rate upconversion, while also compensating for view change latency and scene change latency. One of the goals of the thesis was to implement a frame rate upconversion algorithm that can run on modern GPU architecture.

Another goal of the thesis was to use the algorithm to perform an experiment to see if frame rate upconversion benefits task performance. To fulfill that goal, a first person shooter game like application was designed.

The first part of this section describes the basic format of the experiment. The second part describes the process of building an application which implements an upconversion algorithm. The third part describes how the application was adapted for use in the task performance experiment.

5.1 Experiment Format

It was decided to measure task performance in a setting similar to first person shooter game format. The test subject would by repeatedly presented with a moving target to hit in a 3D scene. The test subject would aim by rotating the view with the mouse so that it's centered on the target, with a help or a reticle. While the test subject is able to rotate the camera, movement is disabled. After the user clicks the mouse button a hit or miss is registered. By averaging multiple shots, a hit rate for the test subject can be obtained.

To measure any effect to the hit rate the upconversion algorithm might provide, three test cases were chosen with different rendering conditions:

- 15 Hz conventional case
- 120 Hz conventional case
- upconverted case

In first and seconds cases a conventional geometry based renderer, or just "conventional renderer" for short, will be used. In one case the scene will be rendered at 15 Hz and in another case at 120 Hz. From now on these cases will be referred to as the conventional 15 Hz case and the conventional 120 Hz case.

The third case, called the upconversion case, will use frame rate upconversion from 15 Hz to 120 Hz. It will also use the 15 Hz conventional renderer as a starting point. In this scheme 1 frame in a sequence of 8 frames will be rendered conventionally and the following 7 will be extrapolated with an image based renderer using 3D warping and mesh based reconstruction. The shorter term "warp renderer" will be used from now on.

It would have been possible to design an algorithm which interpolates new frames between two conventionally rendered frames. But this would have resulted in additional latency of one frame period. As a conscious design choice, extrapolation scheme was chosen for low latency.

Additionally by design the conventional renderer is to render object's momentary velocities to an extra buffer on per-pixel level. This buffer is to be used with the warp renderer in compensating for scene change latency, a novel contribution as far as is known. User input from the mouse will be sampled always before rendering any type of frame with either the geometry or warp renderer. Thus with the frame rate upconversion case full 120 Hz sampling rate is used to determine a new viewpoint for each extrapolated frame. The full sampling rate serves to compensate view change latency with the warp renderer.

In every case, the physical simulation is to be done in lockstep with the geometry based renderer. Thus 120 Hz simulation frequency will be used only with the 120 Hz geometry based renderer, with the two other cases simulating at 15 Hz frequency. The warp renderer will compensate for latency by accounting for spatial changes due to constant velocity in each extrapolated frame. Because physics is still only simulated at 15 Hz, the warp renderer cannot account for forces and acceleration, only velocity.

An assumption is made that these 15 Hz and 120 Hz cases represent the lower and upper bounds respectively for test subject's hit rates. It is hypothesized that hit rates in the upconverted case will lie somewhere between the 15 Hz and 120 Hz conventional cases. The hypothesis can be proven by constructing and disproving two null hypothesis. The first of which is that the upconverted case performs as badly as, or worse, than the conventional 15 Hz case. The second is that the upconverted case performs as well or better than the conventional 120 Hz case.

5.2 Application Implementation

The experiment test application was developed to run on 64 bit Windows 7 operating system. The experiment was performed on a PC computer equipped with a Intel 4790K processor, 8 GiB of memory, AMD Radeon HD 7950 GPU and Asus VG248QE 24" display. While the display supports refresh rates up to 144 Hz, only 120 Hz mode was used. For input a Zowie EC1-A mouse with an Avago 3310 sensor is used. The mouse is configured on firmware level to 800 dpi spatial sampling precision and 1000 Hz temporal sampling precision. While credible measurements are lacking, the sensor has a reputation for linear movement response among online gaming community [64] [65].

5.2.1 Used Libraries

While portability was not considered crucial, in absence of overriding priorities, only portable open source APIs and libraries were used. The first choice that was made was to develop the application in C++11 language with OpenGL API using the modern core profile. Most of actual data processing is done on the GPU with shader hardware, which is in turn programmed in the GLSL language. In the eventual form, the application requires a GPU capable of running OpenGL version 4.1 [66] and GLSL version 4.10.6 [67]. OpenGL Extension Wrangler Library (GLEW) version 1.12.0 [68] was used to manage OpenGL extensions.

The Simple DirectMedia Layer (SDL) library version 2.0.3 [69] was used to provide a basic framework for building the application. It was used for creating windows in the operating system, reading mouse and keyboard input, playing sounds, checking timers, converting text into bitmaps and opening image files.

The tinyobjloader library was used for loading scene objects in Wavefront obj format. A copy was taken at revision a67a60d19fd423449c9e2e6ba6ca2071c1f26d72 from the GitHub repository [70].

While GLSL language has a sufficient mathematics library build into it, C++ language is more limited. To support similar level of functionality in C++ when running CPU code another library was necessary. The OpenGL Mathematics (GLM) [71] library was chosen because it attempts to emulate the same syntax and usage pattern as do the math operations provided by GLSL.

5.2.2 Application Overview

The application was divided into modules with each designed to deal with different application portions. There are 5 modules: SDL, Util, GL, Physics and Game.

The SDL module was designed to act as the main controller of the application. The name is reminiscent of the fact that it's the SDL library that most of the controller functionality is based on. The SDL module initializes all the other modules and maintains the main event loop of the application.

The Util module provides utility functionality to other modules. It handles miscellaneous tasks like reading files, loading images, generating timestamps and rendering text to bitmaps. In addition to the Util module also the Timer class, which is used for high precision time measurement, serves a peripheral utility function.

The GL module is called upon by the SDL module to handle all 3D graphics drawing functionality. The name refers to the fact that the most important job of the module is to manage OpenGL state, though it also handles other graphics related tasks. During initialization it loads all the scene geometry and textures. Care is taken to load all necessary resources into GPU memory during initialization to minimize data transfers between system memory and GPU memory afterwards.

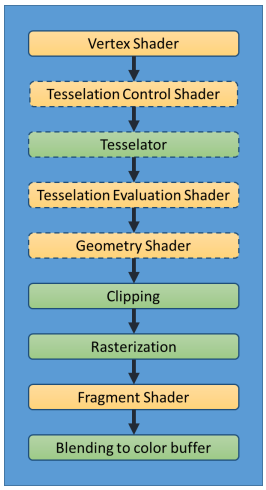

Figure 16: OpenGL 4.1 rendering pipeline overview [66]. Programmable shader stages are in orange. Non-programmable fixed function stages are in green. Stages drawn with dashed borders are optional.

The GL module also initializes the shader programs necessary for drawing the scene. An overview of OpenGL 4 rendering pipeline is displayed in figure 16. An OpenGL shader program is a compilation of all the required and possibly some of the optional shader stages.

Three shader programs were implemented. The first one, the conventional renderer, is used for conventional geometry based rendering. The second one, the warp renderer, is used for image based rendering with 3D warping. The third one, the HUD renderer, is used for drawing a 2D user interface overlay layer.

The Physics module is called upon by the SDL module to simulate scene physics. Originally the scene was designed to have gravity and surface tension forces, which

were approximated with a Runge–Kutta integrator. Later the setting was simplified into a zero-gravity scene with only elastic collisions between the target and the walls. Because of this the integrator was dropped and replaced with an analytic solution to calculating the elastic collisions.

Finally the Game module is tasked with keeping and updating the state of the game which the test subject is conceptually playing. It tracks test phases, records user progress and performs evaluation.

5.2.3 The Conventional Renderer

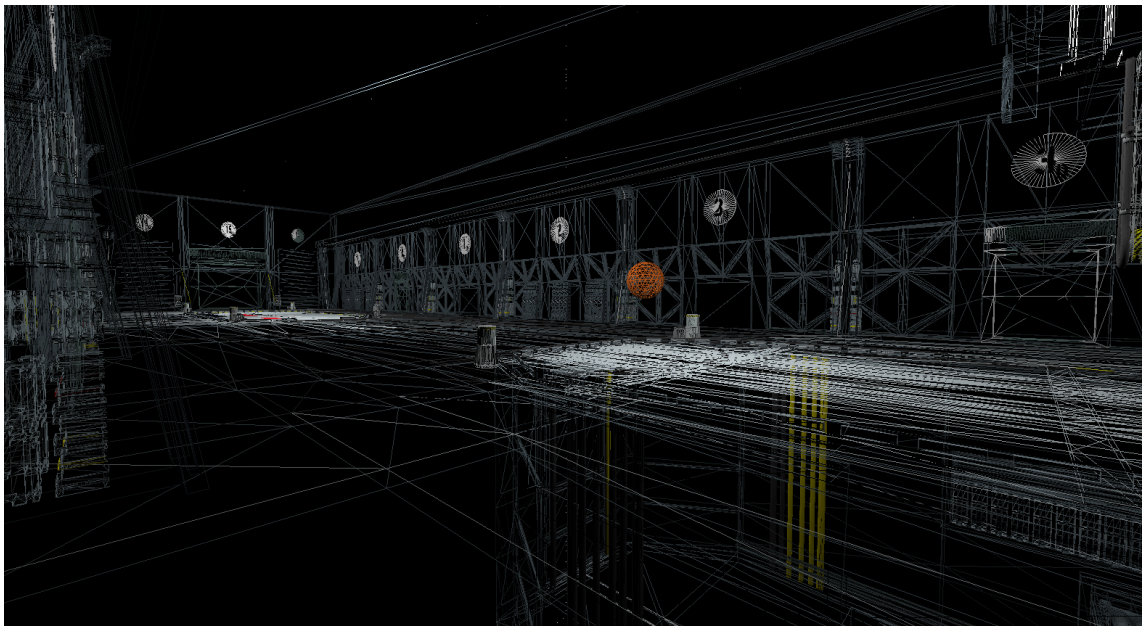

Figure 17: A wireframe drawing of the scene geometry.

The conventional renderer is an OpenGL shader program used for conventional geometry based rendering. It is used in the test application to render the experiment scene based on a geometric description. Figure 17 is a wireframe illustration of the geometric description of the scene from a viewpoint. The exact same viewpoint will be used in subsequent examples.

A simple and common renderer was desired for geometric rendering, so a basic Phong renderer was implemented [44]. This renderer implements only the vertex and fragment shader stages in figure 16. Since the Phong renderer is very common, only an overview is given here. Implementation details with OpenGL core profile may be found from online sources, e.g. [72].

The vertex shader is invoked individually for each vertex for each triangle in the scene geometry and viewing and projection transformation is performed (see equations 2 and 3). After vertex shader the triangles get clipped and finally rasterized by the graphics hardware. After rasterization, fragment shader is invoked. The fragment shader is tasked with determining the colors for each rasterized color fragment. In

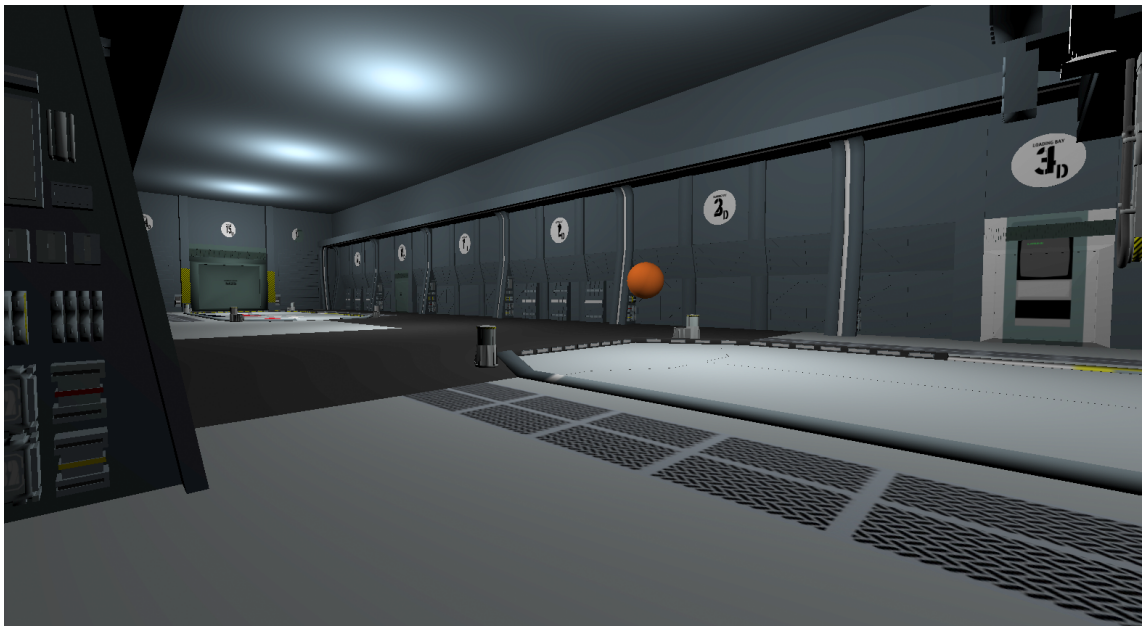

Figure 18: Contents of the color buffer after rendering with conventional renderer.

this case the color is determined using the Phong reflection model [44]. Figure 18 shows the contents of the color buffer after Phong shading.

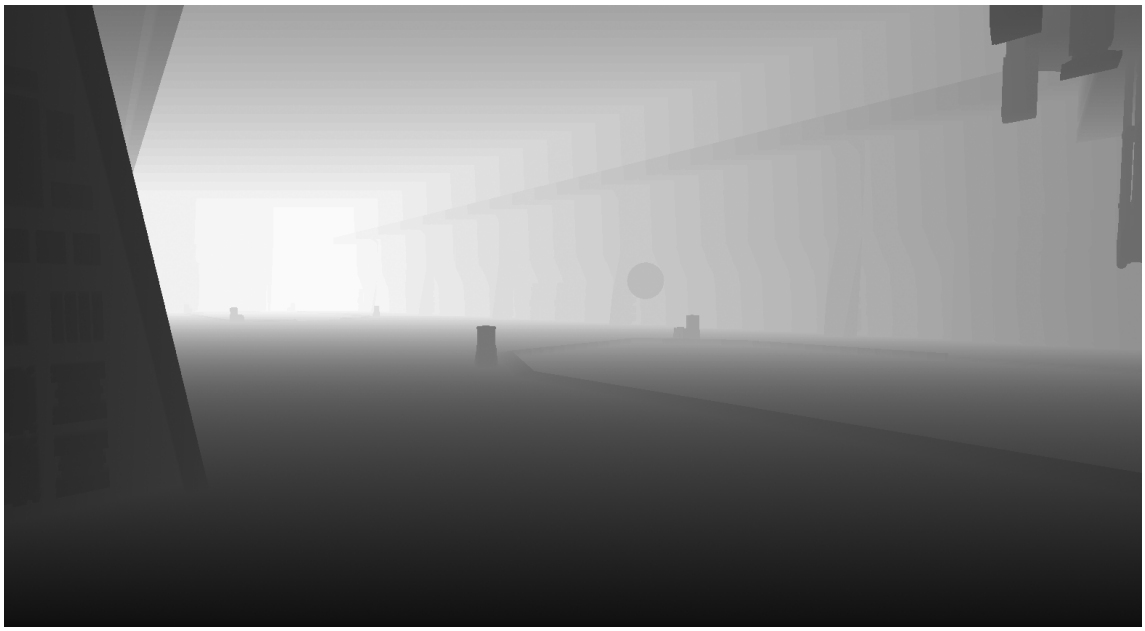

Figure 19: Contents of the depth buffer. Brighter areas correspond to greater Z value, representing greater distance from the projection plane for a given viewpoint.

Each fragment may or may not eventually become a pixel in a color buffer. This is determined by the Z-coordinate from equation 4, used to ensure that only the front most fragments remain visible. The Z-coordinate itself is also saved to another buffer,

the Z-buffer. Figure 19 shows the final contents of the depth buffer after rendering.

While everything done thus far is rather common as intended, there are two special buffers in addition to the color buffer and the Z-buffer which are also filled in by the fragment shader. One buffer is used to store the object id which is currently being drawn by the shader program. This value can later be read back to CPU memory to detect which object was hit by a test subject.

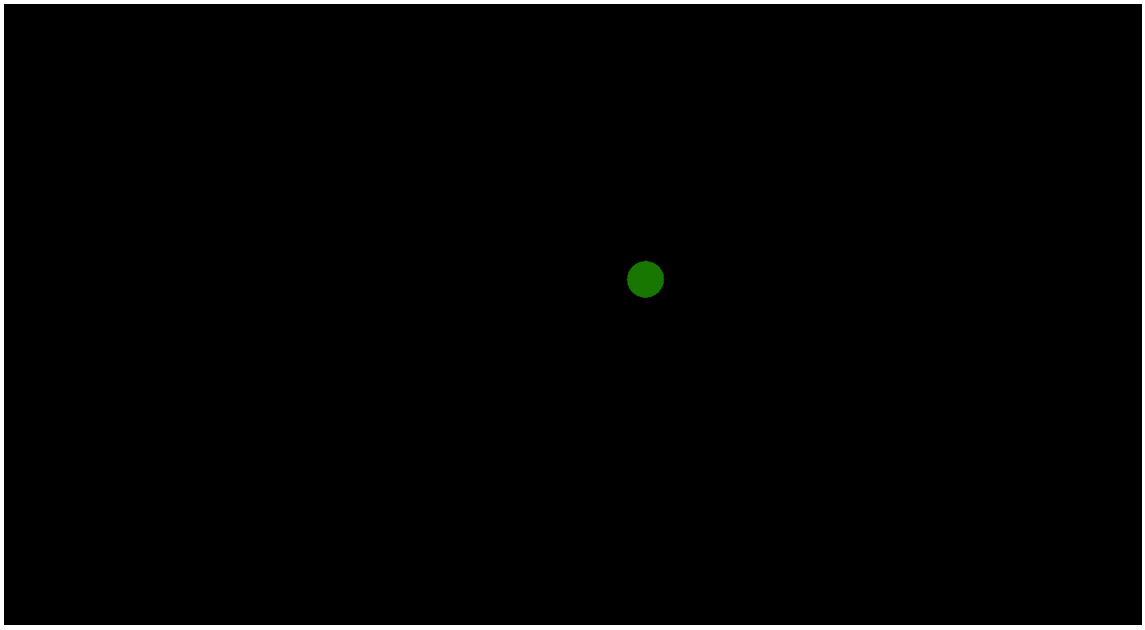

Figure 20: False color image of the velocity buffer. The RGB color values map to x, y, and z world coordinates.

The other special buffer is used for storing the fragments velocity vector in 3D world coordinate system. The velocity value was computed on the CPU by the physics module and is expressed relative to the world coordinate space. While velocity is saved in the buffer separately for each fragment, in practice the value will be constant per each object in the test scene. This is because there are no rotating objects in the test scene. Figure 20 shows the velocity buffer. The only non-black pixels belong to the sphere. This is because it's the only moving objects of that scene. If the sphere were also rotating, the false color image of the sphere would become a color gradient because pixels belonging to the objects would be moving in different directions.

In OpenGL fragment colors may be rendered into an operating system provided frame buffer, which can be directly swapped to the display. In this case a separate offline color buffer is used instead. The reason is that the color buffer and also the associated Z-buffer, hit id buffer and velocity buffer need to be saved. They will be used later as input to the warp renderer. Because of this, after running the shader program, the color buffer is copied into the frame buffer before the frame can be presented on screen.

5.2.4 The Warp Renderer

The warp renderer is a shader program implementing an image based rendering algorithm. A mesh based 3D warp reconstruction method is used. The warp renderer implementation is easier to describe in multiple parts. This subsection explains the initial implementation. The next two subsections describe further optimizations done for the final version which was used in the actual experiment.

In image based rendering, the scene is described only with images, in this case with three OpenGL textures constructed from the output buffers of the conventional renderer: the color buffer, the z-buffer and the velocity buffer. No geometric description of the scene is given. Instead, because a mesh based 3D warp reconstruction method is used, a geometric description of the mesh needs to be given. Note that the warp mesh is rather a geometric description of the original viewport, not the scene itself.

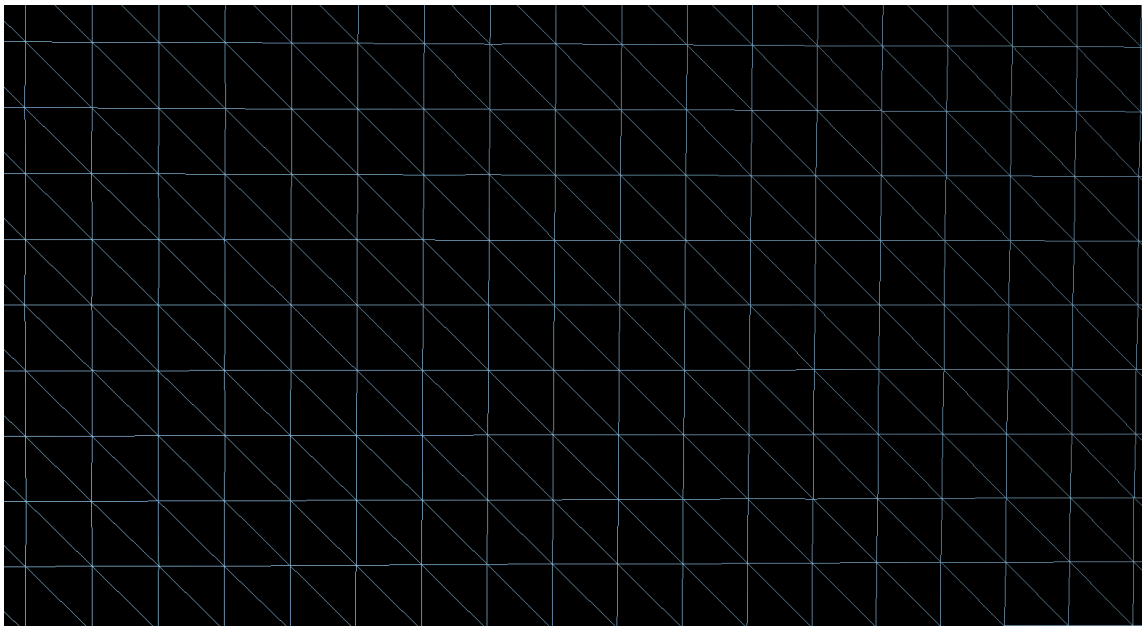

Figure 21: A zoomed in view of the reconstruction mesh. Each vertex represents one of the pixels in the source image.

The reconstruction mesh starts out as a perfectly regular 2D Cartesian grid representing the pixel matrix of the image. Each node in the grid represents a single pixel from where a color value was sampled from. As mentioned before in subsection 4.3.3, this grid can be built out of a set of vertices and triangles. A zoomed in wireframe view of a portion of the grid is shown in figure 21. Each grid node is represented with a vertex. Each group of four neighboring vertices are connected with two triangles. On the abstract level quads would be more natural building blocks of the grid than triangles. Because triangles are the regular building blocks of GPUs though, OpenGL split them into two triangles.

Each vertex is initialized with only one attribute, the 2D position in windows space coordinates of the pixel it represents: the x and y values as in equation 5. The

2D portion of the windows space coordinate is all that is needed because it can be used as a key to read additional color, depth and velocity values from input textures. The grid is constructed and transferred to GPU memory only once during application initialization.

Each invocation of the warp renderer shader program warps this grid according to input frames to extrapolate new frames. The vertex shader, which is executed for each node of the mesh, does the reprojection. The fixed function rasterizer and the programmable fragment shader stages reconstruct the reprojected scene back into an image. The following example is based on the input images in figures 18, 19, and 20.

First in the vertex shader the 2D window space coordinates x and y are used like texture coordinates to read the depth value Z from the depth buffer (figure 19). Using the depth value to augment the nodes coordinates, full window space coordinates for the node as in equation 5 are obtained. Conceptually at this point the mesh is no longer a flat structure, but is warped in the depth direction so that the nodes get snapped to the positions where the surfaces of the original scene geometry used to be. Because this deformation is perpendicular to the projection plane of the original viewpoint, no change would be seen if viewing the grid from the original viewpoint.

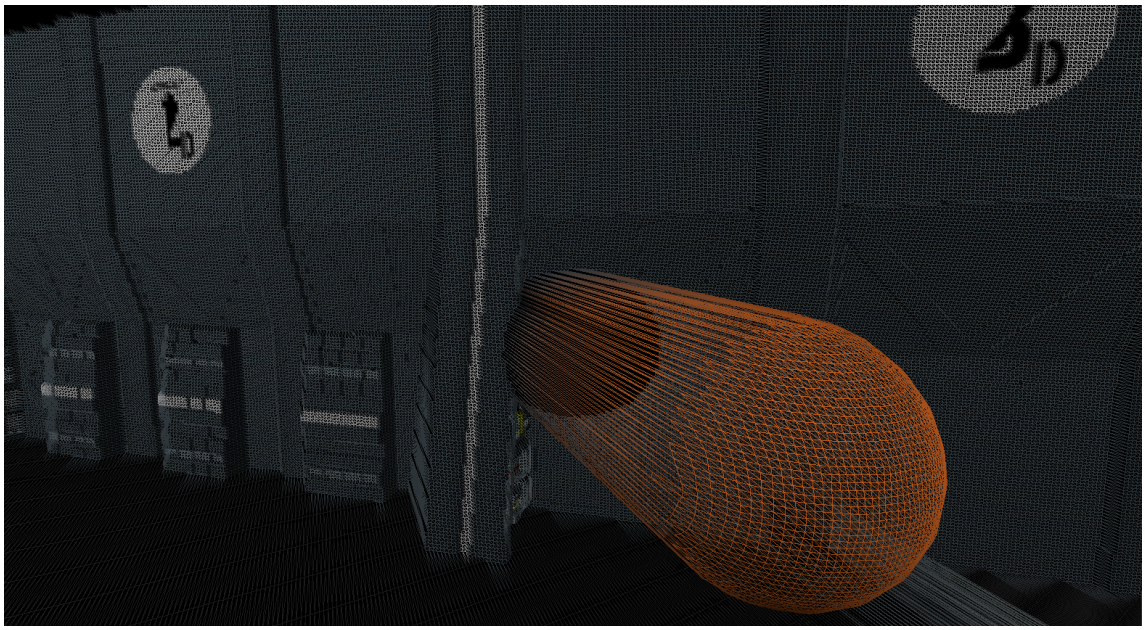

Figure 22: Zoomed-in view of the reconstruction mesh after being augmented with the depth information.

Next using equations 6 and 7 projection and view transformations are reversed to obtain world coordinates for the nodes. Recall that the velocity values in the velocity buffer were also stored relative to world coordinates. Thus this is an opportune moment to factor in spatial changes to those grid nodes that lie on surfaces of moving objects. In figure 22 the mesh is viewed from a zoomed-in viewpoint closer to the sphere than the original viewpoint.

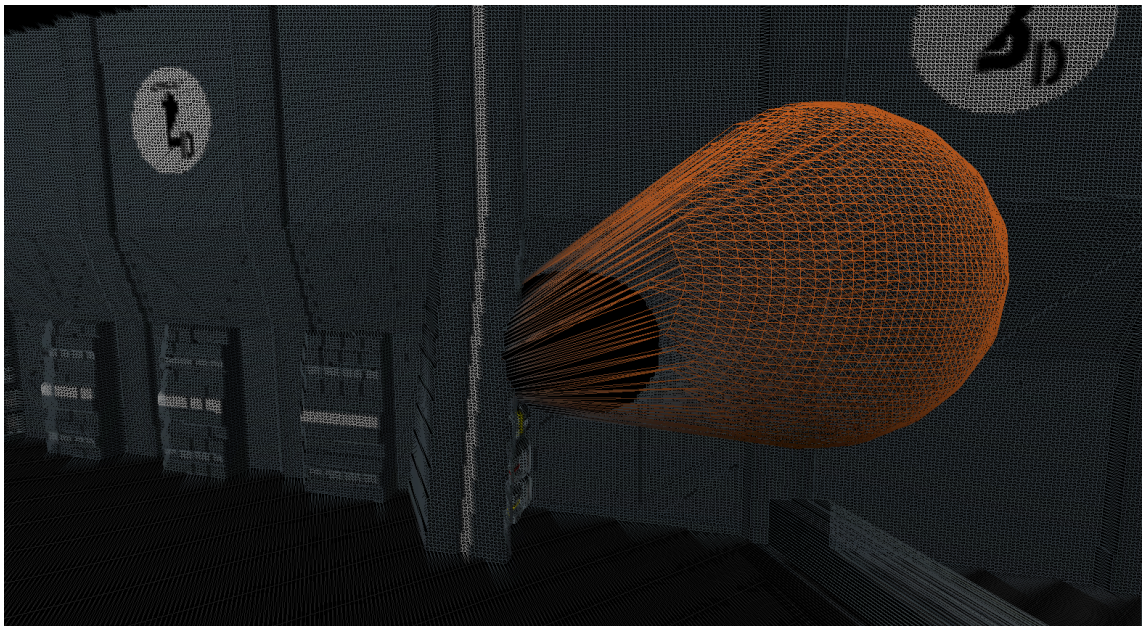

Figure 23: The reconstruction mesh after applying spatial displacement due to velocity.

The 2D window space coordinates x and y are again used as texture coordinates, but this time to obtain the velocity from the velocity buffer (figure 20). The velocity value is a 3D world space vector representing the velocity of whatever surface point the source pixel was sampled from. Displacement due to velocity is computed using equation 11, where x_0 is the nodes position, t is the time elapsed since the source images were rendered with the geometry based renderer and v_{0x} is the velocity. Note that the 1D equation is easily generalized to three dimensions. Vector sums and vector multiplication by scalars are just as valid as the same operations between scalars. The figure 23 is similar to figure 22, but after the displacement effect of velocity has been factored into the mesh nodes. In a sense the mesh in figure 22 corresponds to the time $t_0{=}0$ when the input image was rendered and the mesh in figure 23 some time period t later.

After the displacement is factored into the node's position, it is reprojected to the new desired viewpoint with equation 8, thus obtaining a new set of clip space coordinates. The clip space coordinates are set as the position output of the vertex shader stage.

Of the three textures only the color texture (figure 18) is thus far unused. It will be sampled later in the fragment buffer. For this reason the original 2D window space coordinate is also kept as additional output, so that it will be available later as input in the fragment shader.

After the vertex shader OpenGL takes care of perspective divide (as in equation 9), clipping, and rasterization and finally the fragment shader is executed for each rasterized fragment. Each fragment will belong to some triangle of the warp mesh. Its inputs will be interpolated from the output values of its corner vertices, in this

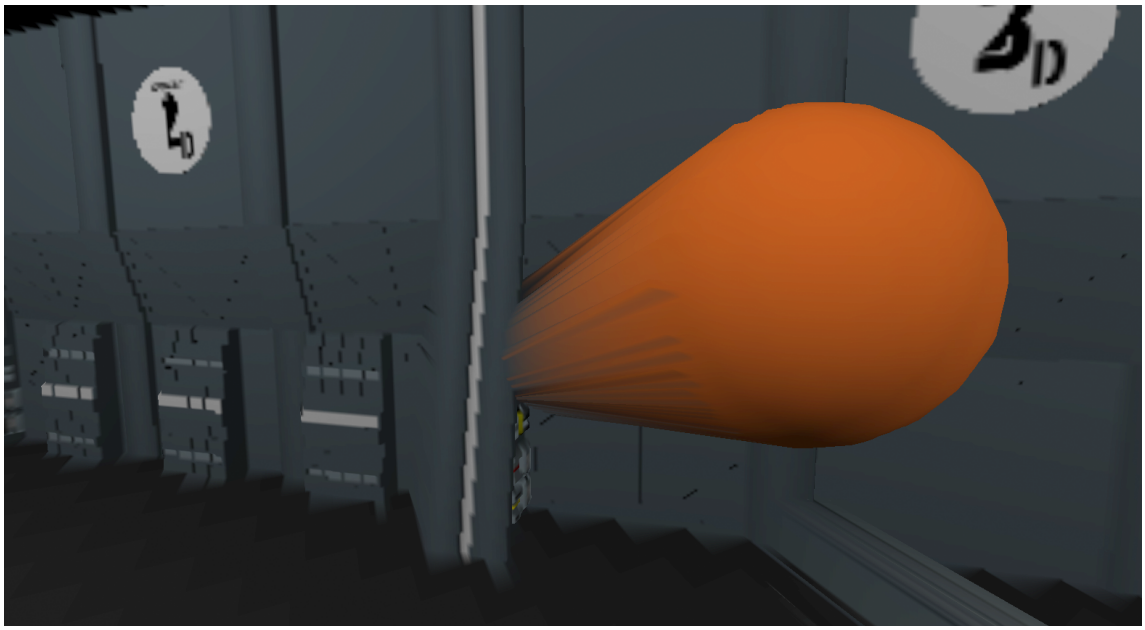

Figure 24: The reconstructed output of the warp renderer.

case the original 2D window space coordinates x and y which were saved. Because the fragment belongs to the reprojected mesh, its position is the reprojected position x' and y'. The original purpose of reprojection was to move each color value in some x and y positions to reprojected x' and y' position. To complete this process, the color texture (figure 18) is sampled at x and y coordinates and the resulting value is set as the fragments color. The reconstructed output image of the warp renderer is shown in figure 24.

5.2.5 Disocclusion Artifact Reduction with Geometry Shader

Refer back to figure 14 in subsection 4.3.5. Mesh based reconstruction produces invalid surface artifacts in disoccluded areas of reprojected images. In a naive implementation the color values of such surfaces are simply interpolated from the colors of all the mesh nodes to which the surfaces belong to. Invalid surfaces are seen in figure 25 as strange smooth gradient surfaces between the sphere and the background.

Note that the image is reprojected rather far from the original viewpoint to exaggerate the disocclusion effect. In practice due to frame-to-frame coherence smaller changes are more common unless a very low source frame rate is used. In [56], Mark et al. the invalid surfaces were not dropped, but rather their color was interpolated only from those nodes which belong to the background. An approach similar in spirit was desired for the warp renderer shader program.

The implementation can be split up into three parts. One method is needed to determine which surfaces are invalid. Once determined to be invalid, another method is needed to determine which of the three corner vertices belong to the background and which to the foreground. Finally the foreground vertices color needs

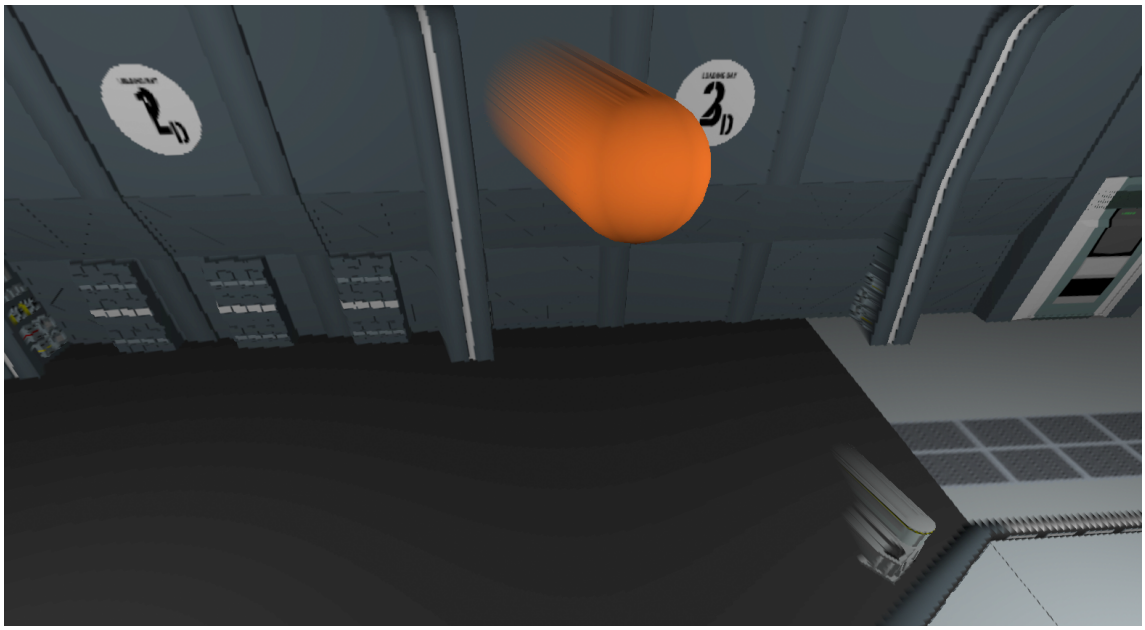

Figure 25: Invalid surface color is interpolated between the background and the foreground.

to be overwritten from the color from the background vertices.

The mesh surfaces, including the invalid surfaces, are implemented as triangles in OpenGL or, from data point of view, as three corner vertices of a triangle. To manipulate whole triangle corner vertices at once in OpenGL, implementation of the optional geometry shader stage is needed (see figure 16). The geometry shader is executed once for each triangle in the reconstruction mesh. It is executed after the vertex shader, so each corner vertices input will be set to equal whatever was output from the vertex shader. In this case each corner vertex will have the reprojected position and the original 2D window space coordinates x and y which were saved separately.

Taking another look at figure 14, we note that invalid surface planes are roughly parallel to lines drawn from the original viewpoint. Similarly their surface normal vectors are roughly perpendicular to lines drawn from the original viewpoint. If the angle between the normal and the line from the original viewpoint is close to 90°, then the surface is detected to be invalid.

Next it needs to be determined which of the three corner vertices belong to the background and which to the foreground. The following algorithm is used. First the vertices are sorted according to distance from eye location (the origin in view space). Second the closest vertex is automatically classified as foreground and similarly the farthest vertex is automatically classified as background. Finally the middle vertex is classified to match whichever of the other two is closer to it.

Now the only thing that remains is to change the color of the foreground vertices to match the background vertices. The actual coloring is performed later in the fragment shader, where the color is looked up from the color buffer using the 2D

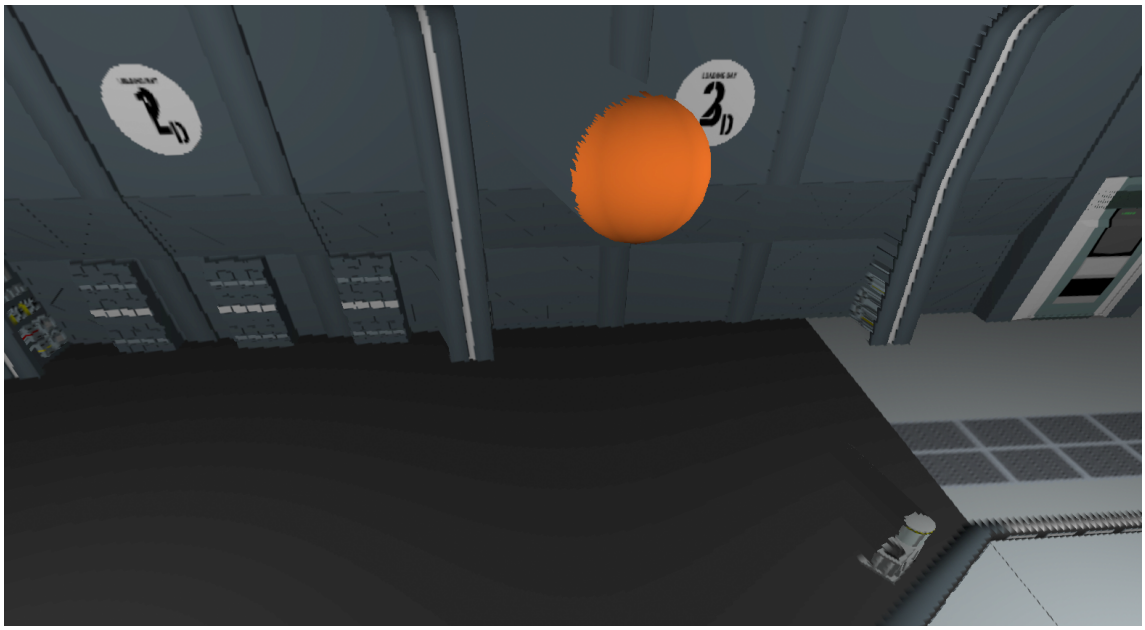

Figure 26: Invalid surface color is interpolated only from background mesh nodes.

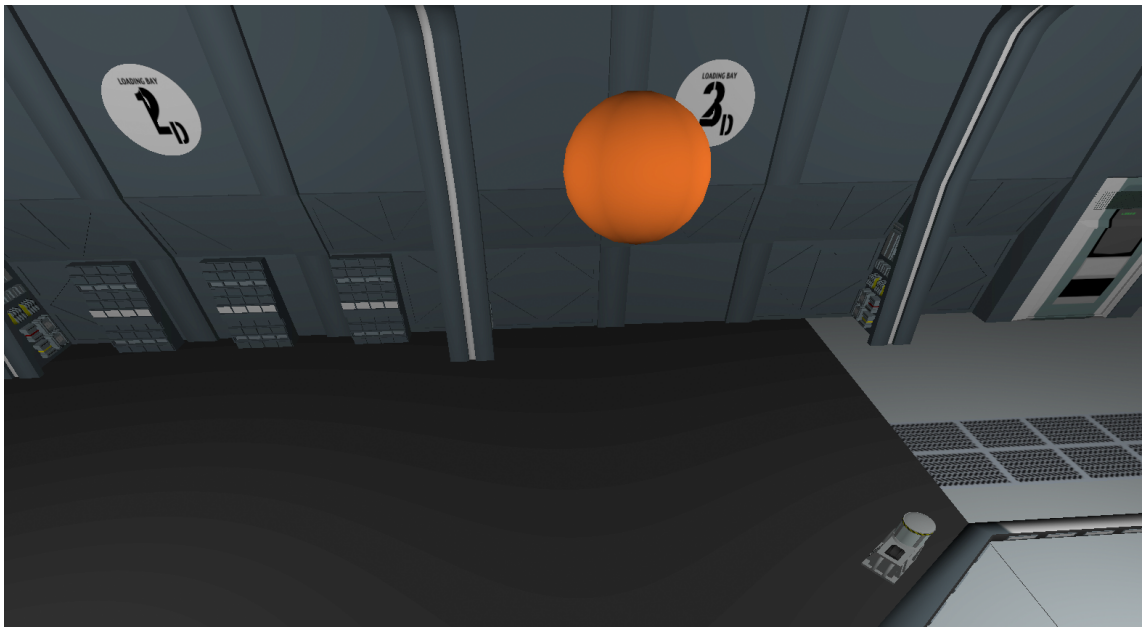

Figure 27: The same viewpoint rerendered with the conventional geometry based renderer for comparison.

window space coordinates. So instead it's the 2D window space coordinates of the foreground vertices which are overwritten with values from the background vertices. The result is shown in figure 26. While the result is far from perfect, on average the disoccluded area is now colored much closer to the true background color in figure 27. And again, the effect here is much exaggerated for the purpose of demonstration.

5.2.6 Sparse Mesh Optimization with Tessellation Shader

The warp renderer turned out to be too slow on the target hardware. It was not able to extrapolate new frames at 120 Hz. This turned out to be due to the very high vertex count because each pixel in the source image contributed a node in the warp reconstruction mesh. The experiment was performed on a 1920x1080 resolution display, resulting in about 2 million vertices and twice as many triangles. As discussed in the subsection 4.3.4, the full density mesh may be approximated with a sparser mesh.

Because it's the depth map that controls how the mesh is warped, lowering the mesh density corresponds in principle to lowering the depth map resolution used in warping. As with any image size reduction, fidelity is only lost in high frequency details. Depth maps on the other hand are mostly smooth gradients (see e.g. figure 19). High frequency information is present mostly only on object boundaries. Mark et al. [56] used a sparser mesh to approximate the full mesh and avoided approximation errors by snapping the mesh nodes to depth boundaries in the depth map (see middle image of figure 13).

It was decided to try using a similar a sparse mesh optimization in the warp renderer. Patch size of 8x8 pixels was found to be optimal after initial testing. Each mesh cell now representing an 8x8 pixel patch in the source image reduces the grids vertex count by a factor of 64. Unlike Mark et al., instead of snapping the mesh nodes to depth boundaries, a novel tessellation based approach was developed. A tessellator is a GPU hardware stage that can subdivide a geometric primitive, a quad in this case, into a number of smaller primitives, also quads in this case (which are later subdivided to two triangles). With tessellation, the grid was redensified locally only in in areas where depth boundaries are found in the depth map. This way the patch may contain more than one depth boundary in a patch without losing significant depth fidelity.

Each 8x8 patch in the depth map was scanned by tessellator for large variations in depth values. The subdivision that the tessellator performs can be controlled separately in both horizontal and vertical orientations. For that reason also the depth data inside the patches is scanned separately in both orientations. Also the patch boundary subdivision is controlled separately from inner subdivision. For that reason it's also necessary to scan the neighboring 4 patches for discontinuities, though only in one orientation.

The process is shown in figure 28. Each vertical and horizontal pair of depth values in the central patch is compared for large differences in depth. The neighboring patches are scanned only in one orientation.

If depth discontinuities are found inside a patch, the tessellator is used to subdivide the grid cell in question back into a full resolution grid. If depth discontinuities are found only in neighboring patches, only the corresponding patch edge gets subdivided. The result of the algorithm can be seen in figure 29. In the figure, around the letter A, most patches were determined to contain no significant depth differences and were not subdivided at all. Near the letter B a patch was fully subdivided because both vertical and horizontal depth differences were found within the patch. Around C

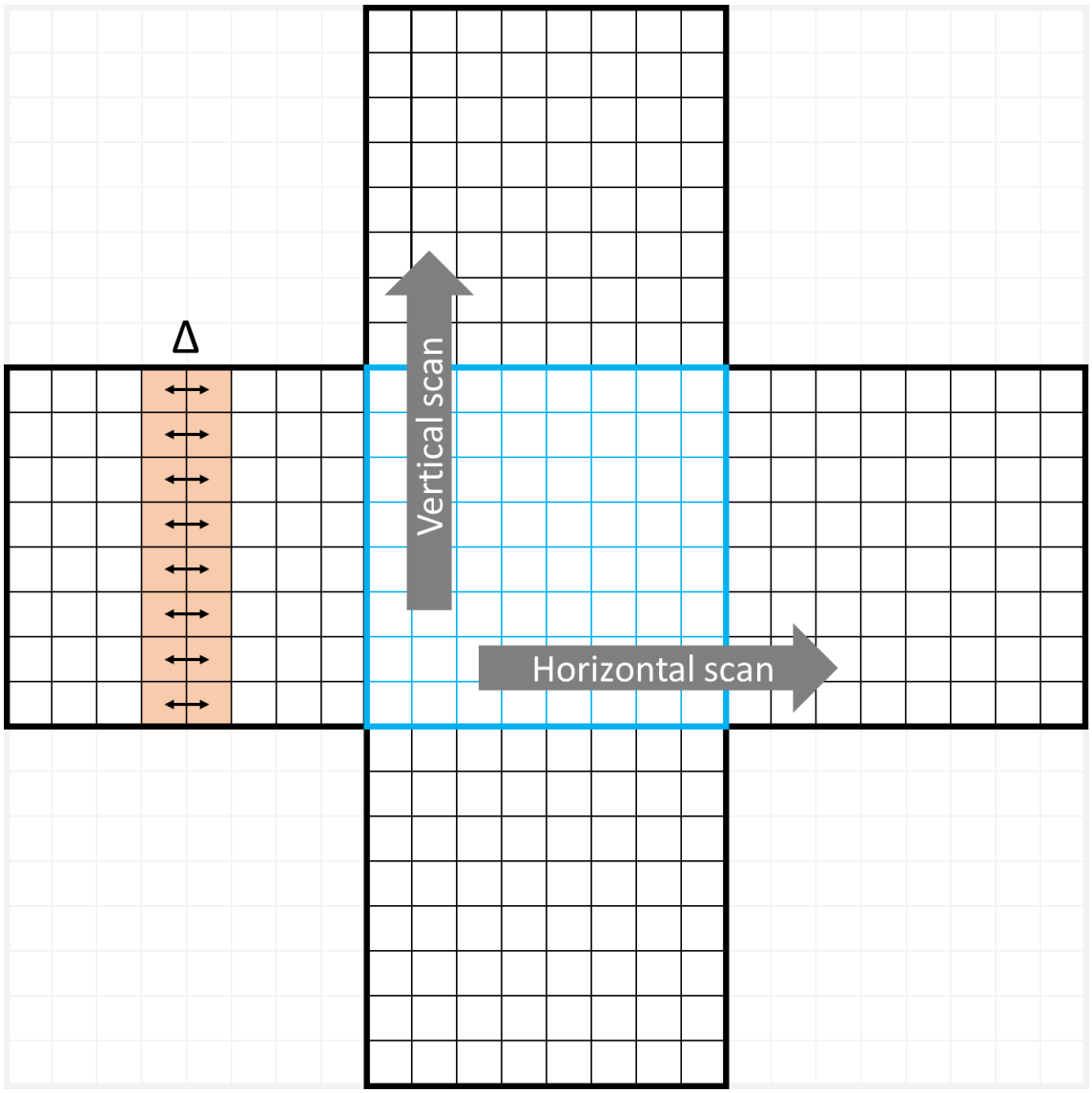

Figure 28: Tessellation control shader algorithm. The thick borders belong to the sparse source mesh patches. Inner squares map to the values of the depth image. The blue central path is the patch currently under evaluation. Additionally four neighboring patches are inspected. The delta symbol represents pairwise depth value subtraction between adjacent depth values.

and D only horizontal or vertical depth differences were found within the patch. The patch under letter E did not contain any significant depth differences and was no subdivided internally. However it's neighboring patches on the right and below were themselves subdivided so the border edges of the patch needed to be subdivided accordingly.

While the implementation of the sparse reconstruction mesh with tessellator is a novelty, it was not central to this thesis. Likely many things could have been optimized better. Such effort was however not meaningful beyond the goal of achieving

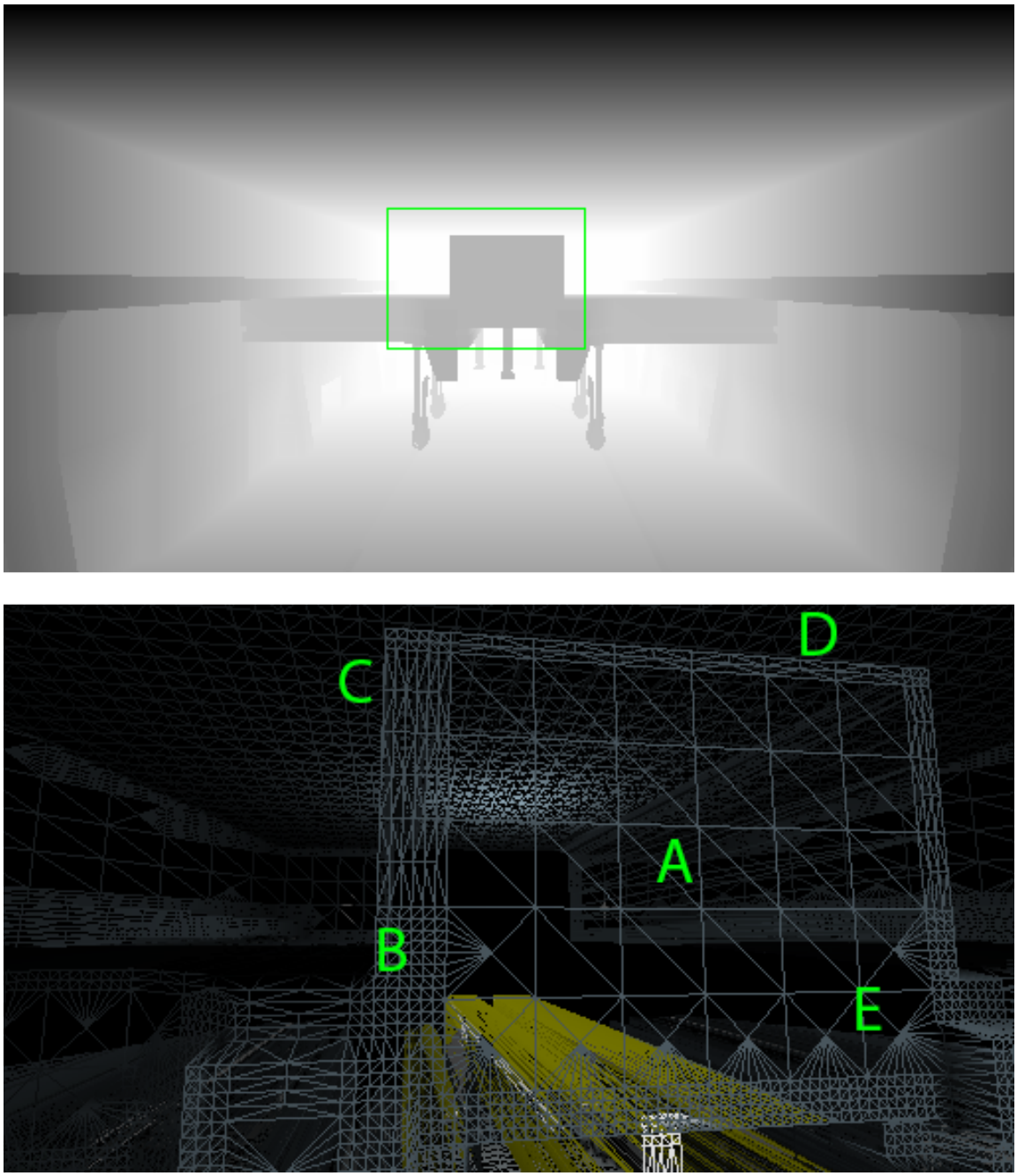

Figure 29: Top: Source depth map with roughly the area of interest highlighted. Bottom: A close up view of a sparse mesh in the area of interest. Letters mark sparse patches that got subdivided in varying ways.

a satisfactory level of performance of 120 frames per second on the target hardware.

5.3 Experiment Virtual Environment

This subsection describes the design of the virtual environment as it is to be presented to the test subjects of the task performance experiment.

5.3.1 Initial Virtual Environment for Technical Development

During technical development, before the details of the experiment were decided, some temporary design choices were made. For the experiment scene the commonly used Crytek Sponza model [73] was used. In the initial implementation, the user was free to move around using a common first person shooter game control method: W, A, S, and D keys for movement on the floor plane and mouse for rotating the camera gaze direction horizontally and vertically. A group of humanoid figures were used as a moving target as they bounced off the floor of the scene while affected by gravity. The physics were simulated with a Runge–Kutta integrator. The user was able to shoot at the target with the left mouse button.

After testing, it was found that there are number of ways to defeat the game, making it too easy to hit the target. First problem was caused by the way bouncing objects behave under gravity. The bouncing target was easiest to hit at its highest elevation. At that point all the kinetic energy has been converted to potential energy and the object comes momentarily to a complete standstill. After practice it was possible to anticipate this location and aim for it instead of tracking the target. The problem was exacerbated by free movement. The user was able to move right next to the bouncing object, wait for it to reach its highest position, and then shoot at a big motionless target.

5.3.2 Designing around the Found Problems

To solve these problems in the final experiment scene implementation a number of changes were envisioned. First, gravity was disabled, and bouncing was made perfectly elastic. Now the target would keep on moving with constant velocity until it bounces off from walls, ceiling, or the floor of the scene. Bouncing would change the direction of movement, but not the speed. This makes it more difficult to cheat. Because the physics of the scene were so much simplified, the Runge–Kutta integrator was replaced with a simple analytic simulator for computing elastic collisions.

Secondly, free movement was disabled. Now user would be able to control only the gaze direction with the mouse, but unable to translate the viewpoint. This change has the tradeoff that disocclusions due to viewpoint translations would no longer occur. However disocclusions caused by the movement of the target itself would still be present, which is expected to be the focal point of the test subjects anyway.

From a neurological point of view, it's desirable to keep the size and speed of the target roughly constant on the test subject's retina. Since free movement was already disabled, it only remains to constrain the targets movement. Towards that end, the target was constrained to bounce in two dimensions on a plane bound by walls, ceiling, and the floor of an enclosing scene. See figure 30 for a floor plan. In the initial condition, before the user manipulates the gaze direction with the mouse, the

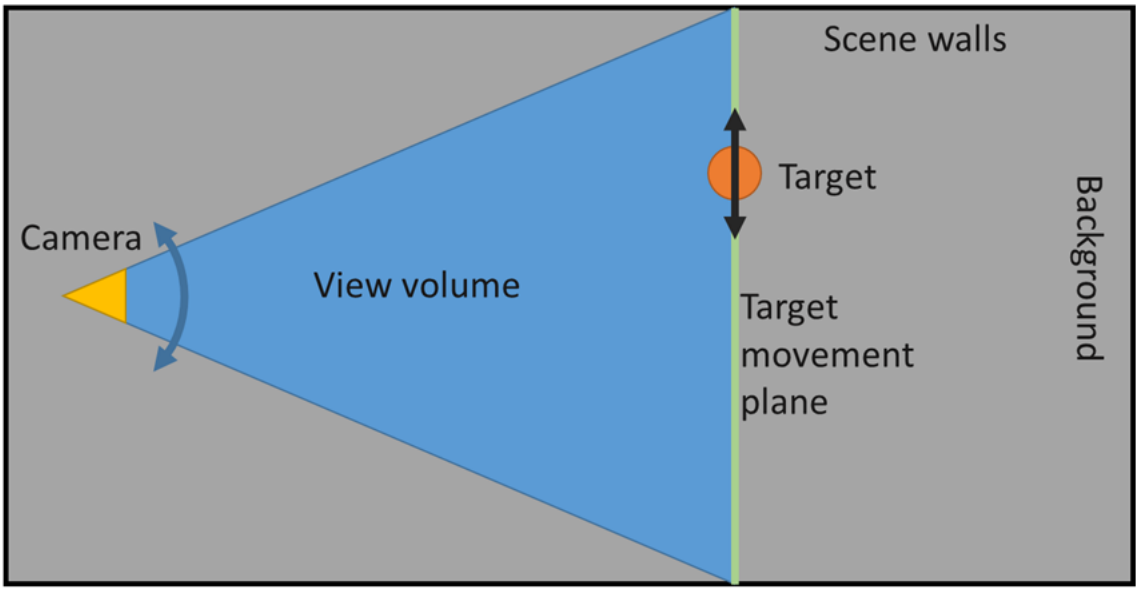

Figure 30: A top down (or sideways) view of envisioned experiment scene.

target movement plane is fitted at such a distance where the view volume and walls of the scene intersect. This ensures that when the target appears, it's always visible from the initial gaze direction. It also ensures that the distance between camera and the target is roughly constant.

Finding a geometric model to replace Sponza was challenging due to narrow constraints. First, since there is no gravity, a space based theme was desired for immersion and plausibility. Further it needed to be a hallway with a cross section of roughly the same aspect ratio as the display (16:9). A model called "Sulaco Hanger" [74] was found to fulfill these requirements after minor stretching. The model was free for non-commercial usage. The target itself was replaced with a simple abstract spherical object.

5.3.3 Game Difficulty

The central goal is to obtain measurements of test subject's hit rate differences under the three test cases. To demonstrate differences the hit rates must not be saturated to 100% accuracy. Similarly, the inverse value, miss rates must not be saturated to 100%. Camera position, target distance and thus the target size on the retina are fixed. Only velocity of the target can therefore be used to adjust difficulty.

$$v_d = v_0 * 2^{\frac{d}{6}} \tag{12}$$

It was assumed that the 120 Hz conventional case will inherently produce higher hit-rates than 15 Hz, i.e. 120 Hz is assumed to be easier. Thus it's necessary to optimize the test difficulty to be suitable for both. The experiment was designed to have a training phase for each test subject, based on the model in figure 31. Initially the target moves slowly enough to ensure a 100% hit rate under every test condition.

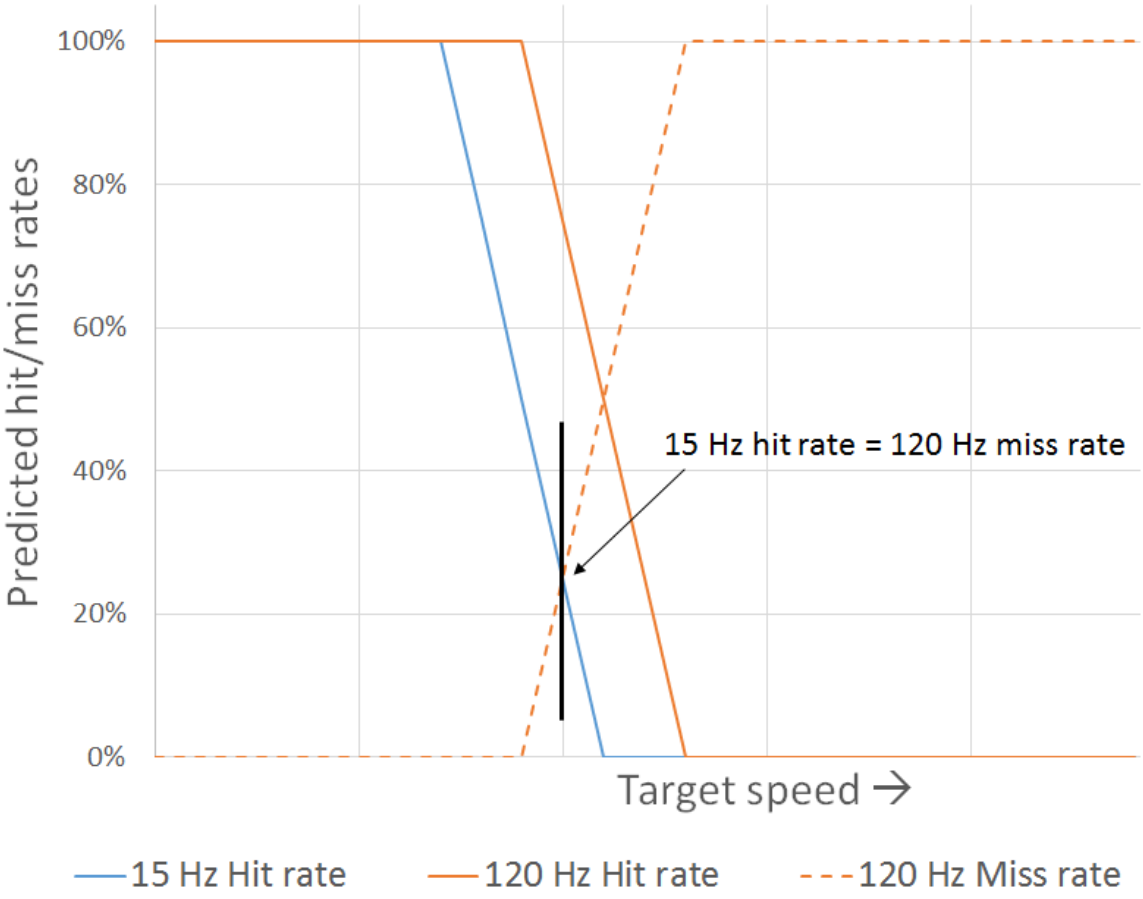

Figure 31: Model for optimizing experiment difficulty.

After observing a number of shots, difficulty is incremented, until hit rates begin to drop. Difficulty is translated to velocity of the target according to formula 12, where d is difficulty. Speed is doubled after each six integer increments of difficulty.

Presumably hit-rates with the 15 Hz renderer drop first. The process of incrementing the difficulty is continued until a suitable difficulty level is reached. Suitable difficulty is determined to have been reached when 120 Hz miss rate exceeds 15 Hz hit rate (see black line in figure 31). The number of shots used for evaluating hit rates varies during training. Initially only a small number of shots are evaluated to allow faster progressing with slow targets. Once performance drops, each difficulty level is evaluated with progressively more shots for better evaluation precision. This allows the test subject to spend less time finding a suitable difficulty level and more time doing the actual experiment.

5.3.4 Final Configuration

During development, it was accidentally found that starting from a high difficulty level, which was previously found suitable for a given person, is too difficult to start with. This suggested that a warm-up period is needed. Also each test subject might not be familiar with first-person shooter game method of controlling the camera with

a mouse. For these reasons even before the training phase, a 5 minute acclimation phase was built in. During this period test subjects were allowed to try shooting the target with the easiest difficulty without anything being recorded or evaluated.

Thus the final version of the experiment application was composed of three phases: acclimation, training and the actual test. A maximum of one hour was reserved for performing each experiment, so these phases needed to be timed accordingly. Each shot was limited to three seconds before timing out with an automatic miss. One second pause was given before a new target was presented. During the pause feedback was given to the user: displaying a freeze frame image of the shot, textual feedback ("HIT", "MISS", "TIMEOUT"), and also distinctive indicator sound was played from speakers.

During the actual test phase, a target was presented for each of the three test cases 225 times in random order, for a total of 675 targets. Since each shot and the associated pause can take a maximum of 4 seconds, the maximum length of the phase is exactly 45 minutes. Due to the varying nature of the training phase, a 12 minute limit was imposed after which the actual test would be forced to begin. Adding these two and the 5 minutes for acclimation goes already over the allotted hour. However it was reasoned that no one would actually need the whole 45 minutes, because that means timing out every shot.

5.3.5 The Experiment Sessions

Naturally as many test subjects as possible is desired. However in context of this thesis it was deemed unpractical to organize more than 20 sessions.

A reward was offered: a choice between a movie ticket and an entry ticket to an annual extracurricular event, which is popular among freshmen of the university. The experiment was advertised through the Facebook page of the said event. Applicants were limited to persons with normal motor functions to make sure they can use a mouse normally. Additionally applicants were limited to persons whom normally use the mouse with their right hand. This was merely a practical matter because no otherwise identical left-hand version of the used mouse was available for the experiment.

A total of 20 test subjects were recruited. The gender distribution was exactly 50%-50%. Additionally 4 persons signed up as backups in case someone cancels. A quiet windowless room was reserved for the sessions. Only the test subject and the supervisor (the author) were present. The full experiment protocol is attached in appendix B.

6 Results

The experiment sessions were held over a span of one week. None of the 20 recruits canceled so no backups were called in. 13 of the test subjects reported having some prior experience with first person shooter games. The test application worked well throughout the experiment and test subjects followed instructions admirably.

Table 1: The raw results from the 20 sessions. The rows are in randomized order (sorted according to the random id). The id is assigned to allow discussing specific test subjects. The hit rates are calculated relative to 225 repetitions of each three test cases.

random person id	120 Hz conventional hit count	upconverted hit count	15 Hz conventional hit count	120 Hz conventional hit rate	upconverted hit rate	15 Hz conventional hit rate
1	170	140	112	0.756	0.622	0.498
2	173	152	119	0.769	0.676	0.529
3	161	144	106	0.716	0.640	0.471
4	127	104	94	0.564	0.462	0.418
5	128	106	75	0.569	0.471	0.333
6	152	123	102	0.676	0.547	0.453
7	169	133	105	0.751	0.591	0.467
8	147	123	108	0.653	0.547	0.480
9	122	107	86	0.542	0.476	0.382
10	104	82	61	0.462	0.364	0.271
11	168	128	114	0.747	0.569	0.507
12	173	156	137	0.769	0.693	0.609
13	91	69	53	0.404	0.307	0.236
14	148	130	89	0.658	0.578	0.396
15	171	161	134	0.760	0.716	0.596
16	137	107	108	0.609	0.476	0.480
17	185	166	122	0.822	0.738	0.542
18	139	119	94	0.618	0.529	0.418
19	170	141	122	0.756	0.627	0.542
20	185	162	161	0.822	0.720	0.716

The results of the sessions are presented in table 1. Inspection of the application event log reveals that unexpectedly high number of test subjects, five in total, exceeded the 12 minute training time limit: subjects 3, 4, 6, 8, and 11. These test subjects were forced to complete the final test phase with possibly lower difficulty than what would have been optimal. In the table it seems their hit rates are not saturated close to 100%. So luckily results from these test subjects don't need to be discarded as invalid.

Additionally the logs reveled an average aiming time of 1.309 seconds for hits and 1.263 seconds for misses. Only 1.6 % of shots ended in a timeout after 3 seconds, which are counted as a miss in table 1.

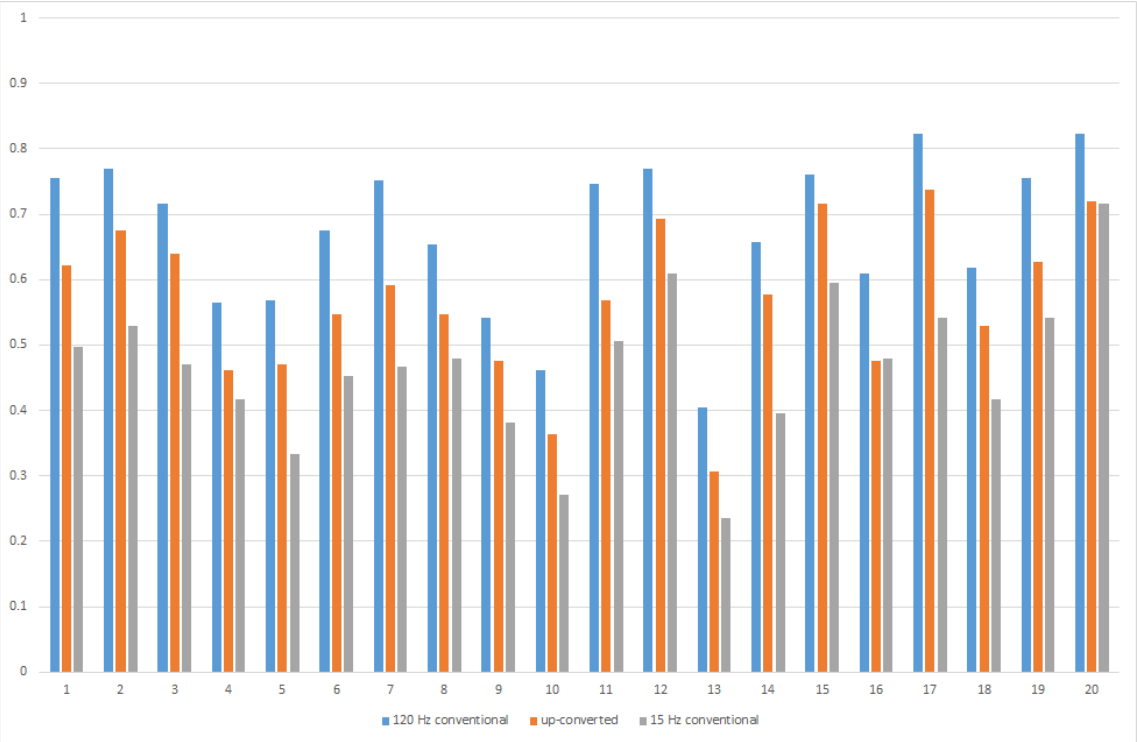

Figure 32: The hit rate results from the 20 sessions.

7 Discussion

The hit rates from table 1 are plotted in figure 32. Only the test subject 16 seems to have a lower hit rate after frame rate is upconverted, though test subject 20 shows only a very minor improvement.

From the data, two questions need to be answered. First, the original hypothesis was defined as: "hit rates in the upconverted case will lie somewhere between the 15 Hz and 120 Hz conventional cases". It needs to be tested. If there is indeed improvement, then the second question becomes: "How much improvement?"

7.1 Sign Testing the Hypothesis

Absolute hit rates of any individual test case are not very interesting. They were obtained from different individuals with varying skill levels. Also difficulty was adjusted to be different for each individual. Instead interesting is the relative performance in the upconverted test case when compared the two reference cases using conventional renderer.

A sign tests is well suited to characterize relative performance. In a sign test one measures results in two cases: a reference case and a test case. The number of measurements where the test case had positive or negative change over the reference case is counted. If the test case offered no change over the reference case, then one may expect there to be roughly equally many positive and negative change observations. The significance of observing unequal amount of positive and negative

changes can be characterized with the binomial distribution.

Table 2: Hit rate differences between conventionally rendered test cases and the upconverted case. Sorted by the difference to 15 Hz conventional case.

random person id	difference between upconverted and 15 Hz conventional hit rates	difference between upconverted and 120 Hz conventional hit rates
16	-0.004	-0.133
20	0.004	-0.102
4	0.044	-0.102
11	0.062	-0.178
8	0.067	-0.107
13	0.071	-0.098
12	0.084	-0.076
19	0.084	-0.129
9	0.093	-0.067
10	0.093	-0.098
6	0.093	-0.129
18	0.111	-0.089
15	0.120	-0.044
1	0.124	-0.133
7	0.124	-0.160
5	0.138	-0.098
2	0.147	-0.093
3	0.169	-0.076
14	0.182	-0.080
17	0.196	-0.084

In table 2 hit rate differences have been calculated for the sign test. Compared to 15 Hz conventionally rendered case, frame rate upconversion showed positive change for 19 out of 20 test subjects. Compared to 120 Hz conventionally rendered case, frame rate upconversion showed negative improvement for each 20 test subjects.

To prove the original hypothesis, two null hypothesis were previously stated. To test the two null hypothesizes, sign tests can be used.

The first null hypothesis was: “upconverted case performs as badly as or worse than the conventional 15 Hz case”. Assuming equal probability of observing positive and negative changes with upconversion, what is the likelihood of obtaining 19 positive changes or more out of 20? The answer is given by a binomial test: likelihood (p-value) is 2.0027160644531247e-05. The binom test was performed with the R language [75]. Clearly, with such a low p-value, the null hypothesis can be rejected at confidence level

in excess of 99.99%. Instead its alternative hypothesis must be true: “upconverted case performs better than the conventional 15 Hz case”.

The other null hypothesis was: “upconverted case performs as well or better than the conventional 120 Hz case”. Assuming equal probability of observing positive and negative changes with upconversion, what is the likelihood of obtaining zero (or less) positive changes out of 20? Again using the binom test, with R language, the likelihood of 9.5367431640624947e-07 is computed. Thus the second null hypothesis is also disproved, and the alternative hypothesis is accepted: “upconverted case performs worse than the conventional 120 Hz case”. Combining the two alternative hypothesis, both lower and upper bounds for the upconverted case are set, implicating that the original hypothesis indeed holds true.

The sign test operates on the differences of the mean values of hit-rates of each individual test subject, where the sample size is 20. As a side note, each hit-rate measurement itself is only an estimate, with 225 samples, of the true hit probability of a given test subject. This error was not explicitly factor into the sign test. It may be argued though that the sample size is already implicitly represented. A smaller shot sample size would have caused larger variance and increased likelihood of some of the signs flipping. Consequently less extreme p-values would have been observed in the sign test and acceptance of the null hypothesizes would have become more likely.

7.2 Improvement Provided by Upconversion

The sign test was used to show that the hit rate indeed improved over conventional 15 Hz rendering. But how much?

Table 3: Quartile boundaries for hit rate differences.

	upconverted hit rate compared to 15 Hz conventional hit rate	upconverted hit rate compared to 120 Hz conventional hit rate
Minimum (0th percentile)	-0.004	-0.178
1st quartile (25th percentile)	0.070	-0.129
Median (50th percentile)	0.093	-0.098
3rd quartile (75th percentile)	0.128	-0.083
Maximum (100th percentile)	0.196	-0.044

In table 3 hit rate differences from table 2 have been split into quantiles. The quantiles were calculated with R. The default R quantile formula (type 7) is used [76].

Absolute values of both medians are roughly equal, indicating that frame rate upconversion increases hit rates to somewhere half way between 15 Hz and 120 Hz conventionally rendered cases. Visual inspection of figure 32 tells the same story. It is also noteworthy that, with the exception of the very slightly negative minimum value, all the percentiles show more than 7 percent point improvements over 15 Hz rendering.

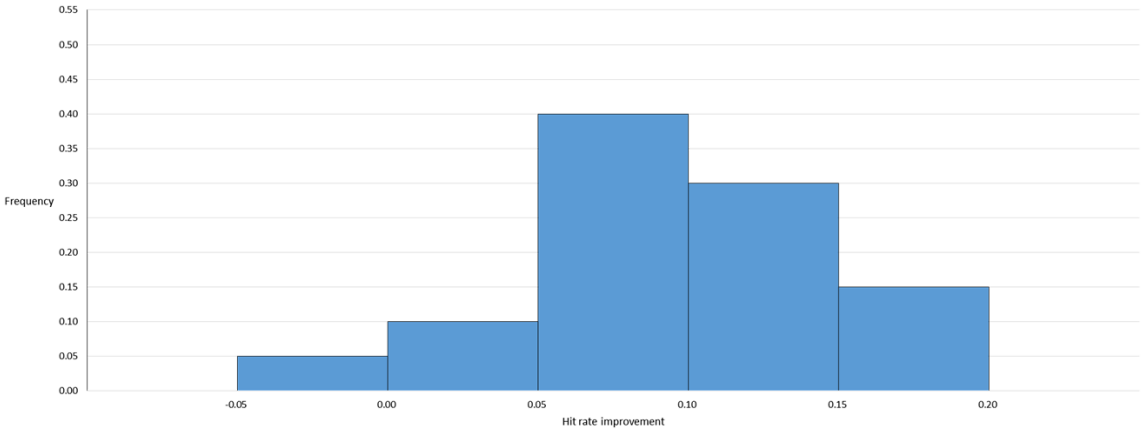

Figure 33: Histogram of upconversion hit-rate improvements compared to 15 Hz conventional rendering.

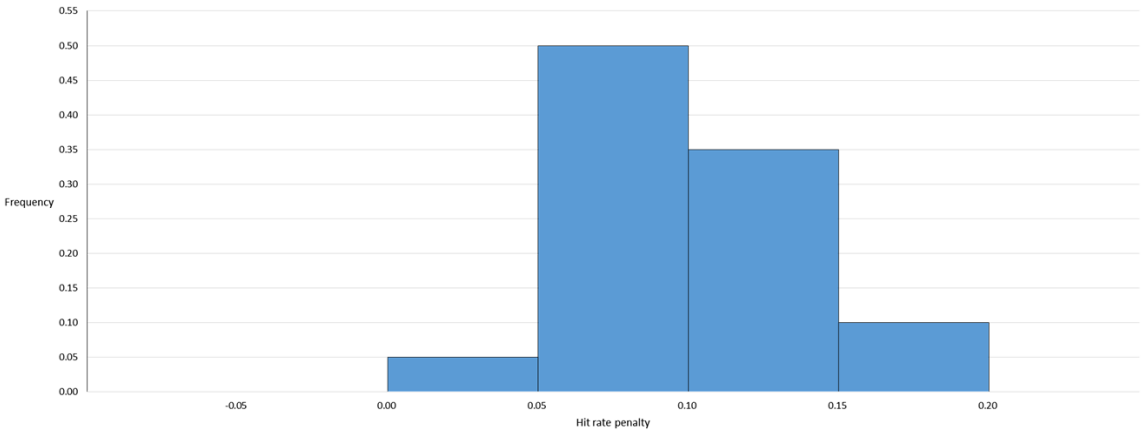

Figure 34: Histogram of upconversion hit-rate reduction compared to 120 Hz conventional rendering.

Figure 33 is a histogram of hit rate improvements over 15 Hz conventional rendering. For majority of test subjects the improvement is somewhere between 5 and 15 percentage points. Figure 34 is a histogram of hit rate penalty (inverse of improvement) compared to 120 Hz conventional rendering. Again, for majority of test subjects the penalty is somewhere between 5 and 15 percentage points.

8 Conclusions

8.1 Conclusions from the Literary Background Study

A review of literary background was done. If was found out that existing theoretical background for frame rate upconversion is not limited to video and television applications. Instead image based rendering, 3D warping, and the associated image reconstruction methods form a solid theoretical basis for frame rate upconversion with live 3D computer graphics. These techniques also allow latency compensation in the rendering pipeline. Two types of latencies had been identified in rendering pipelines: view change latency and scene change latency. Methods to compensate for view change latency had been proposed and developed earlier, but scene change latency received lesser attention.

8.2 A Frame Rate Upconversion Algorithm Was Implemented

An application implementing a simple live frame rate upconversion algorithm was developed. The central piece of it is the OpenGL shader program dubbed the "warp renderer". Though the initial version was slow, after using a sparse reconstruction grid with tessellation, performance was improved. It was able to achieve consistent 120 Hz frame rate on the target GPU while upconverting by extrapolation a 15 Hz source signal originating from a traditional geometry based renderer. The target GPU, an AMD Radeon HD 7950 (also rebranded as Radeon R9 280), is more or less taken to represent modern commodity level hardware.

The warp renderer features view change latency compensation by sampling the user control signal at full rate. By saving an additional velocity buffer from the conventional geometry based render, the warp renderer is able to compensate for scene change latency. This is considered a new contribution.

8.3 A Task Performance Experiment Was Held

An experiment with 20 test subjects was held. The test subjects played a first person shooter type game. In the game they were presented with a bouncing target, which they were required to shoot. Hit rates were measured under three different test cases. Two test cases used conventional geometry based rendering with 15 Hz and 120 Hz signals. A third test case used frame rate upconversion from 15 Hz to 120 Hz.

The analysis of test results showed measurable and considerable improvement in test subject hit rates when using frame rate upconversion with latency compensation. Conversely, test results showed that using full 120 Hz rendering with the conventional geometry based renderer still performs considerably better. It is concluded that frame rate upconversion and latency compensation are promising applications of image based rendering. They are able to enhance task performance in cases where high frame rate rendering with geometry based renderer might be unpractical.

9 Future

Image based rendering, 3D warping, and latency compensation is promising for a number of potential application. Further research into specific application can be beneficent.

9.1 Rendering Techniques Which Blend Pixels

Rendering techniques which blend light contribution from multiple sources into one pixel were not covered in this thesis. Examples include anti-aliasing, transparency, and reflections. These techniques pose a challenge with reprojection, because they can't be reverse mapped to a single point in the virtual world.

Such techniques are commonly used, so they deserve to be consider in detail. One possibility would be to change the GPU rendering pipeline so that the output buffers are not just flat buffers of tuples of color and depth values. Instead the buffer should be able to store lists of tuples, which are blended before displaying on screen, representing samples from different surface points. This way the original information would still be available from each contributing surface point separately. This kind of data structure is similar to a layered depth image [77].

Interestingly, the trend in GPU architectures has been toward increased programmability. E.g. Laine and Karras implemented an experimental software rasterizer [78] which works with GPGPU compute pipeline instead of the graphics pipeline. Unlike the compute pipeline, the graphics pipeline is still partly implemented as fixed function hardware. Such a renderer could certainly be adapted to output fragment lists.

9.2 Dedicated Frame Extrapolation Hardware

It might be beneficent to dedicate a relatively cheap GPU or similar device to frame extrapolation duty. Performance of this GPU needs to be matched only to the display resolution.

At the same time another more powerful GPU would be acting as a rendering server, producing conventionally rendered reference frames, which are themselves never displayed. Such a setup could completely avoid frame pacing problems [79]. The reference frames would need to be transferred between the two GPUs somehow with low latency.

Bibliography

[1] Video Electronics Standards Association, "VESA Releases DisplayPort™ 1.3 Standard," 2014. [Online]. Available: http://www.vesa.org/uncategorized/vesa-releases-displayport-1-3-standard/. [Accessed: 14-Jul-2015].

[2] Video Electronics Standards Association, "DP1.3 and DisplayPort Alt Mode on USB Type-C News Release Media Coverage," 2014. [Online]. Available: http://www.

vesa.org/wp-content/uploads/2014/10/Press-Release-Coverage.docx. [Accessed: 14-Jul-2015].

[3] Martyn Williams, "Sharp to sell the world's first 8K TV starting in October," 16-Sep-2015. [Online]. Available: http://www.computerworld.com/article/2984441/personal-technology/sharp-to-sell-the-worlds-first-8k-tv-starting-in-october.html. [Accessed: 22-Oct-2015].

[4] Oculus VR, LLC, "First Look at the Rift, Shipping Q1 2016," 06-May-2015. [Online]. Available: https://www.oculus.com/en-us/blog/first-look-at-the-rift-shipping-q1-2016/. [Accessed: 30-Jul-2015].

[5] Samsung Electronics Co., Ltd., "Samsung Explores the World of Mobile Virtual Reality with Gear VR," 03-Sep-2014. [Online]. Available: http://www.samsungmobilepress.com/2014/09/03/Samsung-Explores-the-World-of-Mobile-Virtual-Reality-with-Gear-VR-1. [Accessed: 30-Jul-2015].

[6] Sony Computer Entertainment Inc., "SONY COMPUTER ENTERTAINMENT ANNOUNCES 'PROJECT MORPHEUS' - A VIRTUAL REALITY SYSTEM THAT EXPANDS THE WORLD OF PLAYSTATION®4 (PS4™)," 19-Apr-2014. [Online]. Available: http://www.scei.co.jp/corporate/release/140319_e.html. [Accessed: 30-Jul-2015].

[7] HTC Corporation and Valve Corporation, "HTC AND VALVE PARTNER TO MAKE VIRTUAL REALITY DREAM COME TRUE," 01-Mar-2015. [Online]. Available: http://www.htc.com/us/about/newsroom/2015/2015-03-01-htc-valve-partner-to-make-virtual-reality-come-true/. [Accessed: 30-Jul-2015].

[8] Microsoft Corporation, "Windows 10 Announcements: HoloLens, Project Spartan, DirectX 12 and more," 22-Jan-2015. [Online]. Available: http://www.microsoft.com/en-gb/developers/articles/week04jan15/windows-10-announcements-hololens-project-spartan-directx-12-and-more/. [Accessed: 30-Jul-2015].

[9] Business Wire, "FOVE, the World's First Eye Tracking Head Mount Display! Aiming for Global Game and Medical Markets," 25-Jul-2014. [Online]. Available: http://www.businesswire.com/news/home/20140725005247/en/. [Accessed: 30-Jul-2015].

[10] Wired, 10-Oct-2014. [Online]. Available: http://www.wired.com/2014/10/carl-zeiss-vr-headset/. [Accessed: 30-Jul-2015].

[11] Avegant, "Avegant Glyph." [Online]. Available: http://avegant.com/. [Accessed: 30-Jul-2015].

[12] Razer Inc., "Industry Leaders Announce Open Platform for Virtual Reality Gaming," 06-Jan-2015. [Online]. Available: http://www.razerzone.com/press/detail/press-releases/industry-leaders-announce-open-platform-for-virtual-reality-gaming. [Accessed: 30-Jul-2015].

[13] Google Inc., "Google I/O 2014 - Cardboard: VR for Android," 26-Jun-2014. [Online]. Available: https://www.google.com/events/io/schedule/session/603fe228-89c5-e311-b297-00155d5066d7. [Accessed: 30-Jul-2015].

[14] Archos SA, "ARCHOS VR Glasses: Jumping into the mobile virtual reality

world," 16-Oct-2014. [Online]. Available: http://www.archos.com/corporate/press/press_releases/UK_Archos_VR_Glasses_PR_20141016.pdf. [Accessed: 30-Jul-2015].

[15] Starbreeze AB, "Starbreeze launches VR venture with the acquisition of InfinitEye," 14-Jun-2015. [Online]. Available: http://starbreeze.com/2015/06/starbreeze-launches-vr-venture-with-the-acquisition-of-infiniteye/. [Accessed: 30-Jul-2015].

[16] C. Fehn, E. Cooke, O. Schreer, and P. Kauff, "3D analysis and image-based rendering for immersive {tV} applications," *Signal Processing: Image Communication*, vol. 17, no. 9, pp. 705–715, 2002.

[17] John Papadopoulos, "John Carmack Is Pushing Hard For Asynchronous Time Warp On The PC, Best Thing Coming From Mobiles," 21-Sep-2014. [Online]. Available: http://www.dsogaming.com/news/john-carmack-is-pushing-hard-for-asynchronous-time-warp-on-the-pc-best-thing-coming-from-mobiles/. [Accessed: 22-Oct-2015].

[18] Chris Burns, "NVIDIA's Asynchronous Warp made me love Virtual Reality," 19-Sep-2014. [Online]. Available: http://www.slashgear.com/nvidias-asynchronous-warp-made-me-love-virtual-reality-19347228/. [Accessed: 22-Oct-2015].

[19] AMD, "WHITE PAPER, ASYNCHRONOUS SHADERS, UNLOCKING THE FULL POTENTIAL OF THE GPU," 2015. [Online]. Available: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Asynchronous-Shaders-White-Paper-FINAL.pdf. [Accessed: 22-Oct-2015].

[20] J. Ho, "The Samsung Galaxy Note 4 Review," 2014. [Online]. Available: http://www.anandtech.com/show/8613/the-samsung-galaxy-note-4-review/9. [Accessed: 14-Jul-2015].

[21] European Telecommunications Standards Institute, "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio access capabilities (3GPP TS 36.306 version 12.4.0 Release 12)," 2015.

[22] Qualcomm, Inc., "Qualcomm, EE and Huawei Successfully Complete LTE Category 9 Carrier Aggregation Interoperability Testing," 2014. [Online]. Available: https://www.qualcomm.com/news/releases/2014/12/22/qualcomm-ee-and-huawei-successfully-complete-lte-category-9-carrier. [Accessed: 16-Jul-2015].

[23] Ookla, LLC, "Global Download Speed | Net Index from Ookla," 2015. [Online]. Available: http://www.netindex.com/download/map. [Accessed: 16-Jul-2015].

[24] J. B. Kristensen, "Big Buck Bunny, Sunflower version," 2013. [Online]. Available: http://bbb3d.renderfarming.net/download.html. [Accessed: 29-Jul-2015].

[25] G. Van Der Auwera, P. David, and M. Reisslein, "Traffic characteristics of H.264/AVC variable bit rate video," *Communications Magazine, IEEE*, vol. 46, no. 11, pp. 164–174, November 2008.

[26] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the Latency of Cloud Gaming Systems," in *Proceedings of the 19th aCM international conference on multimedia*, 2011, pp. 1269–1272.

[27] G. R. Roelofs, "Compensating for network latency in a multi-player game," US 6475090 B2, 05-Nov-2002.

[28] M. A. Klompenhouwer, "54.1: Comparison of lCD motion blur reduction methods using temporal impulse response and mPRT," *SID Symposium Digest of Technical Papers*, vol. 37, no. 1, pp. 1700–1703, 2006.

[29] D. Burr, "Motion smear," *Nature*, vol. 284. Nature Publishing Group, pp. 164–165, 13-Mar-1980.

[30] J. H. Westerink and K. Teunissen, "Perceived sharpness in complex moving images," *Displays*, vol. 16, no. 2, pp. 89–97, 1995.

[31] F. J. J. Blommaert, "Vision of details in space and time," PhD thesis, Technische Universiteit Eindhoven, 1987.

[32] W. Marinovic and D. H. Arnold, "An illusory distortion of moving form driven by motion deblurring," *Vision Research*, vol. 88, pp. 47–54, 2013.

[33] K. Fukushima, J. Fukushima, T. Warabi, and G. R. Barnes, "Cognitive processes involved in smooth pursuit eye movements: Behavioral evidence, neural substrate and clinical correlation," *Frontiers in Systems Neuroscience*, vol. 7, no. 4, 2013.

[34] M. A. Klompenhouwer, "51.1: Temporal impulse response and bandwidth of displays in relation to motion blur," *SID Symposium Digest of Technical Papers*, vol. 36, no. 1, pp. 1578–1581, 2005.

[35] G. de Haan and M. Klompenhouwer, "An overview of flaws in emerging television displays and remedial video processing," *Consumer Electronics, IEEE Transactions on*, vol. 47, no. 3, pp. 326–334, Aug 2001.

[36] E. Reinhard, E. A. Khan, A. O. Akyuz, and G. Johnson, *Color imaging: Fundamentals and applications*. CRC Press, 2008.

[37] H.-J. Chiu, Y.-K. Lo, J.-T. Chen, S.-J. Cheng, C.-Y. Lin, and S.-C. Mou, "A high-efficiency dimmable lED driver for low-power lighting applications," *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 2, pp. 735–743, Feb 2010.

[38] P. Didyk, E. Eisemann, T. Ritschel, K. Myszkowski, and H.-P. Seidel, "Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays," *Computer Graphics Forum*, vol. 29, no. 2, pp. 713–722, 2010.

[39] H. Okumura and H. Fujiwara, "A new low-image-lag drive method for large-size lCTVs," *Journal of the Society for Information Display*, vol. 1, no. 3, pp. 335–339, 1993.

[40] Mike Seymour, "The Hobbit: Weta returns to Middle-earth," 12-Dec-2012. [Online]. Available: http://www.fxguide.com/featured/the-hobbit-weta/. [Accessed: 01-Oct-2015].

[41] P. Haavisto, J. Juhola, and Y. Neuvo, "Fractional frame rate up-conversion using weighted median filters," *Consumer Electronics, IEEE Transactions on*, vol. 35, no. 3, pp. 272–278, Aug 1989.

[42] P. Shirley, M. Ashikhmin, M. Gleicher, S. Marschner, E. Reinhard, K. Sung, W. Thompson, and P. Willemsen, *Fundamentals of computer graphics*, 2nd ed. Natick, MA, USA: A. K. Peters, Ltd., 2005.

[43] E. Lengyel, *Mathematics for 3D game programming and computer graphics*, 3rd ed. Cengage Learning, 2012.

[44] B. T. Phong, "Illumination for computer generated pictures," *Commun. ACM*, vol. 18, no. 6, pp. 311–317, Jun. 1975.

[45] C. DeCoro and N. Tatarchuk, "Real-time mesh simplification using the gPU," in *Proceedings of the 2007 symposium on interactive 3D graphics and games*, 2007, pp. 161–166.

[46] S. E. Chen and L. Williams, "View interpolation for image synthesis," in *Proceedings of the 20th annual conference on computer graphics and interactive techniques*, 1993, pp. 279–288.

[47] L. McMillan Jr, "AN iMAGE-bASED aPPROACH tO tHREE-dIMENSIONAL cOMPUTER gRAPHICS," PhD thesis, University of North Carolina at Chapel Hill, 1997.

[48] S. Laveau and O. Faugeras, "3-d scene representation as a collection of images," in *Pattern recognition, 1994. vol. 1 - conference a: Computer vision amp; image processing., proceedings of the 12th iAPR international conference on*, 1994, vol. 1, pp. 689–691 vol.1.

[49] H. Hubschman, *Using frame-to-frame coherence in three-dimensional computer animation*. 1981.

[50] H. Hubschman and S. W. Zucker, "Frame-to-frame coherence and the hidden surface computation: Constraints for a convex world," *ACM Trans. Graph.*, vol. 1, no. 2, pp. 129–162, Apr. 1982.

[51] J. Badt Sig, "Two algorithms for taking advantage of temporal coherence in ray tracing," *The Visual Computer*, vol. 4, no. 3, pp. 123–132, 1988.

[52] J. Chapman, T. W. Calvert, and J. Dill, "Exploiting temporal coherence in ray tracing," in *Proceedings of graphics interface '90*, 1990, pp. 196–204.

[53] S. Adelson and L. Hodges, "Generating exact ray-traced animation frames by reprojection," *Computer Graphics and Applications, IEEE*, vol. 15, no. 3, pp. 43–52, May 1995.

[54] L. McMillan and G. Bishop, "Head-tracked stereoscopic display using image warping," in *SPIE proceedings 2409*, 1995, pp. 21–30.

[55] W. R. Mark, G. Bishop, and L. McMillan, "Post-Rendering Image Warping for Latency Compensation," 1996.

[56] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D Warping," in *Proceedings of the 1997 symposium on interactive 3D graphics*, 1997, pp. 7–ff.

[57] eVRydayVR, "Oculus Rift - How Does Time Warping Work?" 19-Apr-2014. [Online]. Available: https://www.youtube.com/watch?v=WvtEXMlQQtI. [Accessed: 07-Nov-2015].

[58] Bob Fisher, "Homogeneous Coordinates," 1997. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/BEARDSLEY/node1.html. [Accessed: 20-Aug-2015].

[59] G. Wolberg, *Digital image warping*, vol. 10662. IEEE computer society press Los Alamitos, 1990.

[60] L. Westover, "Footprint evaluation for volume rendering," *SIGGRAPH Comput. Graph.*, vol. 24, no. 4, pp. 367–376, Sep. 1990.

[61] L. A. Westover, "Splatting: A parallel, feed-forward volume rendering algorithm," PhD thesis, University of North Carolina at Chapel Hill, 1991.

[62] Sir Isaac Newton, *The mathematical principles of natural philosophy*, 3rd ed. Edmond Halley, 1729.

[63] H. D. Young, R. A. Freedman, A. L. Ford, and F. W. Sears, *University physics*, 11th ed. Benjamin-Cummings Pub Co, 2004.

[64] Frederik S, "Mionix NAOS & AVIOR 7000," 2014. [Online]. Available: https://www.techpowerup.com/reviews/Mionix/AVIOR_NAOS_7000/3.html. [Accessed: 19-Oct-2015].

[65] alias "Grim Fandango", "List of mice without acceleration and prediction," 2014. [Online]. Available: https://geekhack.org/index.php?topic=56240.0. [Accessed: 19-Oct-2015].

[66] M. Segal and K. Akeley, "The OpenGL Graphics System: A Specification (Version 4.1 (Core Profile) - July 25, 2010)," 25-Jul-2010. [Online]. Available: https://www.opengl.org/registry/doc/glspec41.core.20100725.pdf. [Accessed: 02-Oct-2015].

[67] J. Kessenich, D. Baldwin, and R. Rost, "The OpenGL Shading Language," 24-Jul-2010. [Online]. Available: https://www.opengl.org/registry/doc/GLSLangSpec.4.10.6.clean.pdf. [Accessed: 02-Oct-2015].

[68] T. O. E. W. Library, "The OpenGL Extension Wrangler Library 1.12.0," 27-Jan-2015. [Online]. Available: http://sourceforge.net/projects/glew/files/glew/1.12.0/. [Accessed: 02-Oct-2015].

[69] Simple DirectMedia Layer, "Simple DirectMedia Layer 2.0.3," 16-Mar-2014. [Online]. Available: http://www.libsdl.org/release/SDL2-devel-2.0.3-VC.zip. [Accessed: 02-Oct-2015].

[70] tinyobjloader GitHub project, "tinyobjloader commit a67a60d19fd423449c9e2e6ba6ca2071c1f26d72," 06-Mar-2015. [Online]. Available: https://github.com/syoyo/tinyobjloader/commit/a67a60d19fd423449c9e2e6ba6ca2071c1f26d72. [Accessed: 02-Oct-2015].

[71] G-Truc Creation, "OpenGL Mathematics 0.9.6.1," 10-Dec-2014. [Online]. Available: https://github.com/g-truc/glm/releases/tag/0.9.6.1. [Accessed: 02-Oct-2015].

[72] Martin Christen, "Per Fragment Lighting," 2007. [Online]. Available: https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/lighting.php. [Accessed: 19-Oct-2015].

[73] Crytek, "Sponza Model," 19-Aug-2010. [Online]. Available: http://www.crytek.com/cryengine/cryengine3/downloads. [Accessed: 21-Oct-2015].

[74] alias "Arjin", "Sulaco Hanger." [Online]. Available: http://www.sharecg.com/v/43031/related/11/poser/sulaco-hanger. [Accessed: 21-Oct-2015].

[75] The R Foundation, "The R Project for Statistical Computing." [Online]. Available: https://www.r-project.org/. [Accessed: 21-Oct-2015].

[76] The R Foundation, "R: Sample Quantiles." [Online]. Available: http://stat.ethz.ch/R-manual/R-patched/library/stats/html/quantile.html. [Accessed: 22-Oct-2015].

[77] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images,"

in *Proceedings of the 25th annual conference on computer graphics and interactive techniques*, 1998, pp. 231–242.

[78] S. Laine and T. Karras, "High-performance software rasterization on GPUs," in *Proceedings of high-performance graphics 2011*, 2011.

[79] Ryan Smith, "AMD Frame Pacing Explored: Catalyst 13.8 Brings Consistency to Crossfire," 01-Aug-2013. [Online]. Available: http://www.anandtech.com/show/7195/amd-frame-pacing-explorer-cat138. [Accessed: 22-Oct-2015].

Appendix A: Source Code

The experiment application is released as open-source under the GPL license (see http://www.gnu.org/licenses/gpl.html). The source code is available from git repository hosted by Bitbucket: https://bitbucket.org/snoukkis/interpolationtestplatform.

Appendix B: Experiment Protocol

Prepare the environment:

1. Attached Zowie EC1-A mouse to the computer.
2. Mouse is configured to purple color coded 800 DPI setting.
3. In Windows Control Panel mouse preferences are configured as follows:
    - Set linear response with no acceleration by unchecking the "enhance pointer precision" checkbox.
    - Set speed multiplier to 1.0 by moving the speed slider to the 6th nudge from the left, i.e. middle position. This makes the speed constant and avoids integer rounding errors.
4. Screen is configured to 120 Hz mode.
5. Disconnect wired and wireless network adapters to avoid interrupting the system with updates etc.
6. Record current date and time.
7. Start the application in production mode.
8. Check sound playback.
9. Did anything unexpected happen?
    - Fix the unexpected problems or cancel the experiment.
10. Take the keyboard out of test subject's reach.
11. Turn phone to silent mode

Test subject arrives:

1. Welcome the test subject.
2. Record the test subject's name.

3. Test subject is asked to put phone and other devices to silent mode or turn them off.
4. Test subject is warned that the experiments lasts maximum 1 hour.
5. Test subject is sat down in front of the computer.
6. Test subject is explained that she/he can first practice inconsequentially for 5 minutes.
7. Test subject is introduced to the application
    - The ball bounces.
    - Your task is to aim and hit the ball.
8. Test subject is explained how to interact:
    - Move mouse to rotate the camera.
        - Ask test subject if she/he prefers to flip the mouse vertical axis (standard game option for old-school first person shooter players used to the joystick style).
        - make note of this
    - Press down the left mouse button to shoot at a sphere.
    - You have only one shot per sphere.
    - Test subject has 3 seconds to aim.
    - Test subject should continue aiming even after pressing down the mouse button.
    - After shooting, the camera freezes for a short moment to get feedback.
    - Also green/red text appears to indicate whether the test subject hit, missed, or timeouted.
    - Additionally a success/fail sound is played.
9. Test subject is advised to practice for 5 minutes.
10. Test subject is asked to make everything comfortable while practicing:
    - Seat should be setup to be comfortable.
    - Mouse and hand should lie comfortably.
    - Sound volume should be comfortably.
    - Ask if the test subject needs anything else to be comfortable before the experiment begins.
11. Test subject is practicing for 5 minutes.
12. Test subject is asked if she/he is comfortable.
13. She/he is informed that the experiment ends when the sphere no longer reappears.
14. The test subject is explained that the supervisor has earplugs on and to speak up loudly if there are any problems or when the experiment ends.
15. She/he is reminded to hit the ball **as much as possible**.
16. The enter key is pressed to begin the experiment and test subject is told: “the experiment has started, good luck”.
17. Experiment continues for max 57 minutes... While test subject is hitting the ball, keep timed log of anything out of ordinary.

18. Test subject receives the agreed compensation.
19. Test subject is politely thanked very much for her/his contribution to science.